Graph Theoretic Approach for Constrained Error Control Codes

by

Sangseob Song

A Thesis
Presented to the University of Manitoba
in Partial Fulfillment of the Requirement for
the Doctor of Philosophy
in
Department of Electrical and Computer Engineering

Winnipeg, Manitoba

August 1990

ISBN 0-315-71847-1

GRAPH THEORETIC APPROACH FOR CONSTRAINED ERROR CONTROL CODES

BY

SANGSEOB SONG

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

DOCTOR OF PHILOSOPHY

Contents

List of Figures

List of Tables

Acknowledgement

I would like to thank my supervisor Dr. Ed Shwedyk for sharing his knowledge by introducing me to this subject and for constant support and encouragement. I am grateful to Dr. J. K. Wolf, Dr. H. C. Finlayson, Dr. H. K. Kim, Dr. H. C. Card for serving on my committee and reading my thesis. Special thanks go to Dr. B. Kocay, many of the key ideas came out of discussions with him. Also thanks are due to Chinsong for drawing numerous figures; to Joongsun for typesetting tables; to Yuns who helped with programming, gave good criticism and ideas; to Glenn for good discussions and for taking the trouble to understand the details; to Dave who made "Sun-life" easier; to Jay for his joy and humour. Special thanks go to my colleague professors, Chonbuk National University, for their love and support for the past 4 years. Thanks for the financial support to this research are due to Korea Electric Association, to University of Manitoba Fellowship, and also to NSERC. This thesis is dedicated to my family Yonghee, Dohae, Jammy, and new born Edwina, for years of love and support.

Abstract

A new technique for the construction of constrained error control codes is presented. The main emphasis is on runlength limited (RLL) codes. RLL codes, commonly known as (d, k) constrained codes, where d is the minimum run of zeroes and k the maximum, are routinely used in magnetic and optical digital storage systems. Traditional RLL codes leave the task of combatting errors to separate error control codes. The combined construction of forward error correction (FEC) and RLL codes has received attention recently. Here a *graph search* technique motivated by the work of Ferreira to construct combined RLL/FEC codes is discussed. Ferreira's Hamming map is viewed as a subgraph embedded in a *distance graph* and graph algorithms are exploited to find Hamming maps if they exist. The graph search algorithm chooses a set of 2^n constrained codewords at a time represented by *matching* edges of the *distance graph* and rearranges them into a Hamming map by an efficient combinatorial technique termed *canonical ordering* algorithm. Being able to find many Hamming maps, this graph search technique has been used to find good Hamming maps which result in a large free distance and small k-constraint. This technique is applicable to dc-balanced codes as well. Several new codes are presented and compared to known codes.

Chapter 1

Introduction

Runlength limited (RLL) codes are utilized by a class of input restricted binary symmetric channels such as digital data storage systems, whether magnetic [14, 34, 35, 46, 52], or optical [31]. RLL sequences possess the property that the allowable number of consecutive "0" symbols in a sequence, called zero runlength, is constrained between two predefined parameters d and k, where $0 \leq d < k$. The significance of these constraints are discussed later.

Since the avoidance of undesirable sequences was the primary concern of RLL codes, the error handling aspect has not been reflected in traditional construction techniques of RLL codes. For this reason RLL codes are often referred to as recording or modulation codes to emphasize that their roles are different from that of an error control code. Error correction capability is achieved by a separate code specifically designed for this purpose. Therefore a practical system employs two "concatenated" codes with the error control code being the "outer code" and the RLL code being the "inner code." For the error correction task, a vast variety of efficient linear block codes and convolutional codes have been developed [20, 22, 60]. Excellent discussions on error correction codes can be found in many texts such as Blahut [9], Lin and Costello [48], etc., or in a collection of papers edited by Berlekamp [8].

The combined construction of error correction and RLL code as an entity has received attention starting from the mid 1980's. The so called RLL forward error correction (RLL/FEC) codes try to achieve two goals simultaneously: the given runlength constraints and also that the code should be capable of correcting errors. This thesis considers the problem of combined construction of RLL/FEC codes and tries to answer how to construct them in some systematic fashion. In the following sections, a brief historical survey is presented, the thesis goal is elaborated, and the thesis outline is described.

1.1 Earlier RLL codes

The importance of runlength constraints is due to the particular way information is ascribed to the behavior of certain physical devices with two states. In the writing (sending) mode one causes a transition from one state to the other to occur at specified time intervals. In the reading (receiving) mode the time intervals between transitions are measured. This measurement is made in terms of integral multiples of a unit of time called a *clock unit.* If there are $l+1$ clock units between two transitions, then the symbol 1 is assigned to the first clock unit and 0 to the remaining l. The appearance of $[d, k]$ constraints with this scheme is natural. In order to detect transitions properly and to avoid potential intersymbol interference, transitions cannot occur too close together, so a minimum allowable time, say t_{min}, between them is prescribed. If $t_{min} = d + 1$ clock units, then one gets a lower bound d for the runlength of 0. On the other hand, transitions provide synchronization information to the clock which is an imperfect device. Clocks drift and lose power to discriminate the number of clock units between transitions which are far apart. Requiring that transitions be separated by no more than $k + 1$ clock units places an upper-bound k on the run of 0.

When the RLL codes are used in magnetic recording devices, the transitions are magnetic ones occurring in circular tracks on disks. The quantity t_{min} can be equated to the reciprocal of maximum density of magnetic transitions (flux changes) on a track, and the following formula can be a measure of recorded *information density* E as the amount of information per unit time (unit distance along a track).

$$E = E[d, k] = p/q \, \frac{d+1}{t_{min}},$$

where p/q is the code rate.

Since RLL codes, in their general form, were pioneered by Franaszek [24, 25] in the late 1960's, a considerable amount of engineering and mathematical literature has been written on the subject. In 1970, Tang and Bahl [58] described many important properties of $[d, k]$ constrained sequences and also devised several RLL codes implemented in block code form. Survey papers given by Siegel [56], and Kobayashi [41] describe properties and design techniques for the practical construction of RLL encoders and decoders.

One of the techniques used in RLL code construction is the sequence state method originated by Franaszek [24, 25]. It was improved by various forms of look-ahead (future-dependent) techniques. Look-ahead (LA) techniques have been studied by many authors including Patel [52], Jacoby [34], Franaszek [26], Cohn and Jacoby [14], Lempel and Cohn [46], and Jacoby and Kost [35]. One objective of look-ahead codes, as described in [46], is to overcome the codeword length restriction encountered in the sequence state methods. Look-ahead encoding rules allow several alternative encodings for given input words. The alternative chosen to encode the input word depends on a finite number of future input words (look-ahead), as well as finite look-back.

Originally these techniques were to a large extent ad hoc. Recently (1983), Adler, Coppersmith, and Hassner have placed them on a firm mathematical basis with the sliding block algorithm [1]. The sliding block algorithm, derived from the branch of mathematics known as symbolic dynamics, represents a theoretical breakthrough in code construction, with significant practical implications. For the first time, the algorithm provides an explicit formula, backed by rigorous mathematical proofs, for the construction of *simple, efficient* RLL codes with *limited error propagation.* The method incorporates many of the key ideas which appear in the preceding work of Franaszek, Patel, Jacoby, Cohn and Lempel, generalizes them and makes precise the construction steps. It is an extension of a coding theorem of Marcus [50], who used techniques developed independently of the recording terminology. This procedure successfully constructs a code of any rate $p/q \leq C$, where C is the channel capacity of runlength constrained systems. It later was extended to handle variable-length sequence state methods [2]. Some related results have followed subsequently [4, 37, 51].

Meanwhile, Burkhardt [11], and Wood and Peterson [62] have proposed the use of maximum likelihood sequence estimation (MLSE) on the magnetic recording channels to detect constrained code sequences. In MLSE detection, the free distance of the code plays an important role in the performance evaluation of code. Defined as the minimum value of the Hamming or Euclidean distances between all possible sequences of codewords, the free distance determines the error performance in MLSE detection [21]. Close examination of traditional RLL codes in use today, however, invariably reveals unity free distances, and thus leave the task of combatting errors to separate error control codes.

1.2 Goal of Thesis

As stated earlier, the combined construction of error correction and RLL code as an entity has begun to receive attention starting from the mid 1980's. It is commonly believed that the combined RLL/FEC codes are possible at the expense of coding rates. For these types of codes, two rather different approaches have been taken. One method exploits the rich distance structure of known error control codes such as linear convolutional codes to construct RLL/FEC codes. The idea is to lower bound the Euclidean distance of a trellis code by the Hamming free distance of a linear convolutional code. This approach has been used by Wolf and Ungerboeck [61], and Calderbank, Heegard, and Lee [13]. Wolf and Ungerboeck described $[0, k]$ error correcting codes for partial response channels with discrete-time transfer functions $(1 \pm D)$, where D is the unit-delay operator. They demonstrated that using well known convolutional codes in conjunction with precoding before the partial response channel yields the desired trellis codes. The trellis codes prevent unlimited runs of identical signals at the channel output and have a minimum squared Euclidean distance between channel output sequences which is bounded below by the Hamming free distance, d^H_{free}, of the convolutional code. Calderbank, et al. [13] describe binary convolutional codes for a partial response channel with the discrete-time transfer function $(1 - D^N)/2$, where the minimum squared Euclidean distance between channel outputs corresponding to distinct inputs is bounded below by the free distance of a convolutional code which they called *magnitude code*. To limit the runlength of the channel output, they employ a coset of the binary convolutional code (called *sign code*) to generate channel inputs.

The other approach is exemplified in papers by Ferreira, et al. [17, 18, 19], Lee and Wolf [44], and Song and Shwedyk [57], who try direct construction of runlength constrained codes. Ferreira, Hope, and Nel described a rate 4/8 runlength constrained error correcting code with free distance three [18]. Lee and Wolf introduced two simple $[d, k]$ trellis codes of free distance three which consequently have error correction capability [44]. Ferreira [19] developed a tree search algorithm to find a so called Hamming distance preserving mapping, an ordered set of constrained sequences, which substitute codewords of a punctured convolutional code [12] to obtain an RLL/FEC code.

However, the construction of combined error correction and recording code as an entity is generally a young problem. As the goal of this thesis, a *graph search* technique is investigated. The *graph search* technique is motivated by the work of Ferreira [19]. Ferreira's Hamming map is viewed as a subgraph embedded in a *distance graph* and graph algorithms are exploited to find (virtually) all Hamming maps if they exist. Therefore this approach enables one to single out the optimum Hamming map(s) which result(s) in the greatest free distance with the least k-constraint.

As shall be seen, the problem of searching for a desirable mapping is nothing more than a maximum matching in a graph and that this technique is applicable to dc-balanced codes as well.

1.3 Outline of Thesis

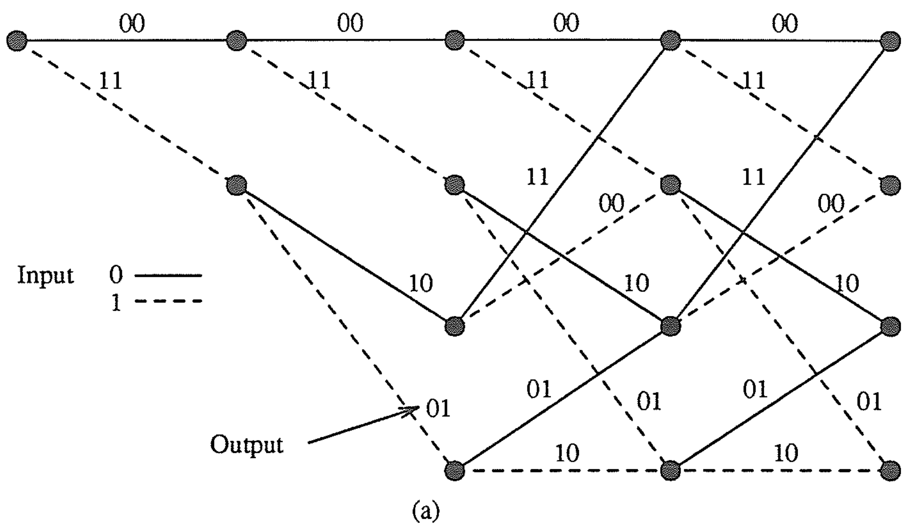
Chapter 2 refines the necessary conditions developed by Ferreira for a Hamming distance preserving map to exist. The refinements lead to a reduction in the computation. The Hamming map search is formulated as a graph theory problem, and the linear graph structure is analyzed in terms of *matching* and *adjacency*. Chapter 3 describes the algorithms for listing the maximum matchings and arranging the constrained sequence pairs (matching edges) into a Hamming map. A summary of the $[d, k]$-constrained codes as well as some dc-balanced codes are presented in Chapter 4. The codes are compared to known codes. Chapter 5 presents the conclusions and recommendations.

Chapter 2

Hamming Maps

One technique of RLL/FEC code construction is to translate the codewords of a linear convolutional code into constrained codewords such that pairwise Hamming distances are preserved [19]. In this way, the free distance of the constrained trellis code is inherited from (bounded below by the free distance of) the base convolutional code. Fig. 2.1 illustrates this.

The trellis in Fig. 2.1(a) depicts a well-known linear convolutional code of rate $k/n = 1/2$ with constraint length $\nu = 2$—correspondingly having 4-states—and free distance $d_f^u = 5$. Note that the same symbol k is used to denote either the maximum zero runlength or the number of information bits of convolutional codes. The meaning should be clear from the context. The $[d, k] = [1, 5]$-constrained codeword labels in Fig. 2.1(b) are concatenable without violating runlength constraints. As can be checked easily, Hamming distances between any two constrained codewords in each frame in Fig. 2.1(b) are bounded below by the corresponding counterparts of Fig. 2.1(a) and so are the free distances. The actual free distance d_f^c of the trellis code in



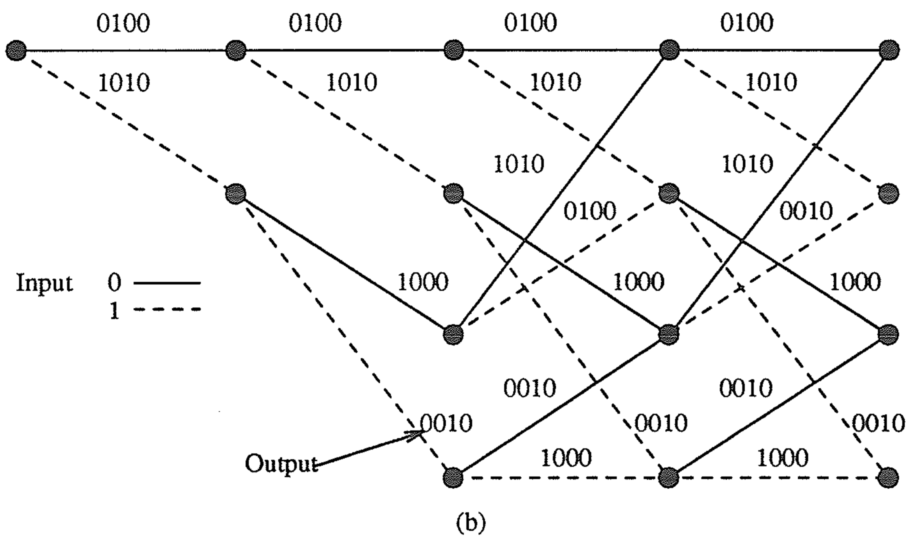


Figure 2.1: (a). A rate 1/2 convolutional code. (b). A rate 1/4 $[d = 1, k = 5]$ RLL constrained code.

Fig. 2.1(b) is six, yielding a $[d,k] = [1,5]$, rate 1/4 RLL/FEC code. The correspondence between constrained and unconstrained codewords is called a *Hamming distance preserving map*, or simply *Hamming map.*

In the following, the precise definition of the problem and terminology is given first. A refinement to the necessary conditions for the existence of a Hamming map is made and compared to those presented by Ferrreira. The chapter concludes by formulating the determination of the Hamming map as a graph problem.

2.1 Problem definition

Consider the RLL/FEC code scheme shown in Fig. 2.2. The runlength limited forward error correction code consists of a rate $r_u = k/n$ binary linear convolutional code followed by a mapper. The mapper translates an n-bit convolutional codeword into an l-bit $[d,k]$-constrained codeword, $l > n$, such that the Hamming distance is preserved under this mapping. The resulting trellis code constructed in this way will have rate $r_c = k/l$ and free distance $d_f^c \geq d_f^u$, where d_f^u is the free distance of the base convolutional code.

Denote the finite set of integers from 0 to $2^n - 1$ by U, where each integer i is represented as an n-bit binary sequence u_i. Note that codewords of a rate k/n binary convolutional code are contained in the set U.

$$U = \{u_0, u_1, \ldots, u_{q-1}\}$$

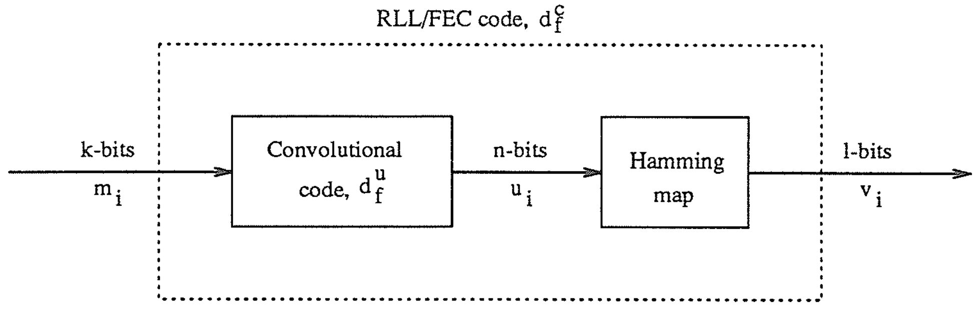


Figure 2.2: Schematic representation of an RLL/FEC code.

where $q = 2^n$. Let C denote the set of d-constrained binary codewords of length l.

$$C = \{c_0, c_1, \ldots, c_{N_d(l)-1}\}$$

where $N_d(l)$ is the cardinality of set C. Here only the d-constraint is specified. The maximum runlength k is determined after the convolutional codewords are replaced with the constrained codewords.

Assuming that the last d bits of a codeword are the symbol "0" to ensure concatenability, the number $N_d(l)$ of all d-constrained binary codewords of length l can be computed with a slight modification by the recursive formula of Tang and Bahl [58] as:

$$N_d(l) = \begin{cases} 1 & \text{for } 1 \leq l \leq d \\ N_d(l-1) + N_d(l-d-1) & \text{for } l > d. \end{cases} \tag{2.1}$$

This formula is based on the following observation. When the first bit of an l-bit sequence is zero, the next $l-1$ bits is any d-constrained sequence of length $l-1$. If the first bit is nonzero, the next d bits are all zeros followed

Table 2.1: Number of binary d-constrained sequences with last d-zero bits.

$d:l$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	2	3	5	8	13	21	34	55	89	144	233	377	610
2	–	1	2	3	4	6	9	13	19	28	41	60	88	129
3	–	–	1	2	3	4	5	7	10	14	19	26	36	50
4	–	–	–	1	2	3	4	5	6	8	11	15	20	26
5	–	–	–	–	1	2	3	4	5	6	7	9	12	16

by any d-constrained sequence of length $l-d-1$. Table 2.1 shows the values of $N_d(l)$ for various parameters.

To transform a rate $r_u = k/n$ linear convolutional code with free distance d_f^u into a rate $r_c = k/l$ constrained trellis code with free distance d_f^c, where $d_f^c \geq d_f^u$, it is necessary that:

$$N_d(l) \geq 2^n \tag{2.2}$$

It is easy to see that $l > n$. For efficient code rates only mappings with the smallest l satisfying the inequality (2.2) are investigated.

To determine the mapping associate a $q \times q$ *Hamming distance matrix* D_u with the unconstrained set U:

$$D_u = [d_{ij}^u] = [d(u_i, u_j)], \; i,j = 0,1,\ldots,q-1.$$

Similarly, an $N_d(l) \times N_d(l)$ Hamming distance matrix D_c is associated with the d-constrained set C:

$$D_c = [d_{ij}^c] = [d(c_i, c_j)], \; i,j = 0,1,\ldots,N_d(l)-1.$$

Example: When $d = 1$ and $n = 2$, U has 4 elements and C has $(N_1(4) = 5)$ elements.

$$\begin{aligned}
\{u_0, u_1, u_2, u_3\} &= \{00(0), 01(1), 10(2), 11(3)\} \\
\{c_0, c_1, c_2, c_3, c_4\} &= \{0000(0_H), 0010(2_H), 0100(4_H), 1000(8_H), 1010(A_H)\}
\end{aligned}$$

$$D_u = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 1 \\ 2 & 1 & 1 & 0 \end{pmatrix} \qquad D_c = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 \\ 1 & 0 & 2 & 2 & 1 \\ 1 & 2 & 0 & 2 & 3 \\ 1 & 2 & 2 & 0 & 1 \\ 2 & 1 & 3 & 1 & 0 \end{pmatrix} \tag{2.3}$$

It is convenient to define an index (slot) set $I = (0, 1, \ldots, q-1)$, $q = 2^n$. Without loss of generality, it will be assumed that $u_i = i$, $\forall u_i \in U$. In this sense, the symbols U and I will be used interchangeably.

Definition 2.1 (Hamming map) *Let V be a q-tuple of constrained sequences:*

$$V = (v_0, v_1, \ldots, v_{q-1}), \; v_i \in C, \; \forall i \in I.$$

Then an one-to-one mapping $f : U \to V$ is called a Hamming map if the Hamming distance is preserved under this map, i.e.,

$$d(v_i, v_j) = d(f(u_i), f(u_j)) \geq d(u_i, u_j), \; \forall i, j \in I. \tag{2.4}$$

This inequality can be written in matrix form as:

$$D_v \geq D_u,$$

where D_v is a $q \times q$ distance matrix with $d(v_i, v_j)$ as its ij-th entry.

Table 2.2: Computational requirement for an exhaustive search, $d = 1$.

k	n	l	$q = 2^n$	$N_d(l)$	$r_u = k/n$	$r_c = k/l$	Computations
1	2	4	4	5	1/2	1/4	120
2	3	5	8	8	2/3	2/5	40,320
2	3	6	8	13	2/3	2/6	51,891,840
3	4	7	16	21	3/4	3/7	4.257579×10^{17}
4	5	8	32	34	4/5	4/8	1.476164×10^{38}
4	5	9	32	55	4/5	4/9	4.911185×10^{50}

The problem is to find *all* Hamming maps f or q-tuples V provided a Hamming map exists. After all Hamming maps are collected, one can single out the optimum Hamming map(s) which yields the greatest free distance with the least k-constraint. To do this with an exhaustive search would require $\begin{pmatrix} N_d(l) \\ q \end{pmatrix} \cdot (\ q\ !\ )$ trials—all q-combinations out of $N_d(l)$ and $q!$ permutations for each combination. Table 2.2 lists the required computation for various n when $d = 1$ and the base convolutional code has rate $r_u = k/n = k/(k+1)$. As can be seen, the computation grows rapidly even for parameters of moderate values.

Ferreira [19] developed a so called **tree search algorithm** to find a single Hamming map, if one exists. The q-tuple V that is found is dependent on how the elements of set C are chosen. Before applying the tree search algorithm, three *necessary conditions* were applied to the chosen elements to determine whether this could result in a map. The algorithm and necessary conditions are reviewed next.

2.2 Tree Search Algorithm

The tree search algorithm finds a suitable vector V by choosing the first element $v_0 \in C$ arbitrarily, next the second element v_1, and so forth. At depth h, one needs to look back to assure that $d^v_{hj} \geq d^u_{hj}$, $0 \leq j \leq h-1$. This is done by computing a metric function $t(h, c_i)$ for each constrained l-tuple:

$$t(h, c_i) = \prod_{j=0}^{h-1} [s(d^v_{hj} - d^u_{hj})], \tag{2.5}$$

where $s(i)$ is the unit step function with $s(0) = 1$. Assign any one of the constrained l-tuples with $t(h, c_i) = 1$ to v_h and proceed to depth $h + 1$. Repeat this procedure until at some depth $z < q$, $t(z, c_i) = 0$ for all i. If this happens, **retreat** one step and assign some other c_i with $t(z-1, c_i) = 1$ to v_{z-1}, then again proceed forward to depth z.

The case of $U = (0, 1, 2, 3)$ is illustrated in Fig. 2.3. It produces a vector $V = (0100(4_H), 0010(2_H), 1000(8_H), 1010(A_H))$ which has the required distance matrix: *i.e.*, $d^v_{ij} \geq d^u_{ij}$ for all i, j.

$$D_v = \begin{pmatrix} 0 & 2 & 2 & 3 \\ 2 & 0 & 2 & 1 \\ 2 & 2 & 0 & 1 \\ 3 & 1 & 1 & 0 \end{pmatrix} \qquad D_u = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 1 \\ 2 & 1 & 1 & 0 \end{pmatrix} \tag{2.6}$$

Thus the map $f : U \rightarrow V$ transforms the rate 1/2 linear convolutional code into a rate 1/4 nonlinear constrained trellis code whose free distance is bounded below by the free distance of the underlying convolutional code.

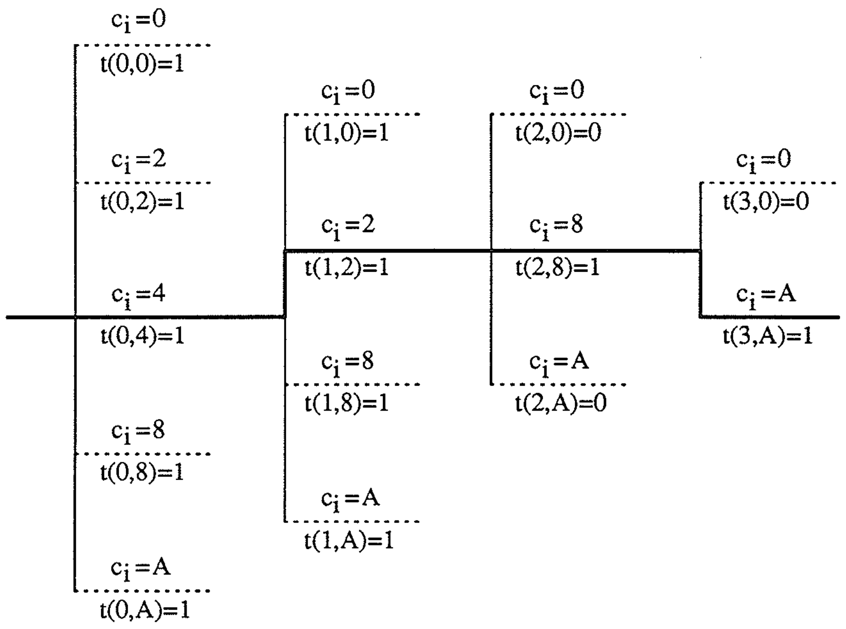


Figure 2.3: Tree search for desired permutation of ($d = 1$)-constrained codewords of length ($l = 4$) [Ferreira, 1989].

2.2.1 Refinement to the Necessary Conditions for the Existence of a Hamming map

Because the tree search can require a vast amount of computation, Ferreira performed several preliminary tests to determine whether a Hamming map exists. The three necessary conditions NC1, NC2, and NC3 are restated without proof as they follow immediately from the inequality (2.4). A refined necessary condition is described that leads to a reduction in the computation.

NC1. Matrix sum test

For a set C of constrained sequences to produce a Hamming map, it is necessary that

$$\sum_{i=0}^{N_d(l)-1} \sum_{j=0}^{N_d(l)-1} d^c_{ij} \geq \sum_{i=0}^{q-1} \sum_{j=0}^{q-1} d^u_{ij} = n2^{2n}/2 = nq^2/2 \tag{2.7}$$

NC2. Matrix row sum test

For a constrained sequence c_i to be a member of a Hamming map, it is necessary that

$$\sum_{j=0}^{N_d(l)-1} d^c_{ij} \geq \sum_{j=0}^{q-1} d^u_{ij} = n2^{n-1} = nq/2 \tag{2.8}$$

Note that NC1 is applied to a set C and NC2 is to a sequence c_i in the set C. Also note that NC2 is stronger than NC1. To discuss NC3 and the refinement of it, the following terms are introduced.

Definition 2.2 (Degree sequence) *Let B denote the set of L-bit sequences. For a sequence $v \in B$, denote the set of sequences at Hamming distance r, $0 \leq r \leq L$, from v as:*

$$S(v;r) = \{w \in B \mid d(v,w) = r\}$$

The cardinality of the set $S(v;r)$ is represented by the symbol δ_v^r and called the r-th degree of v. The set Δ_v of sequences δ_v^r is called the degree sequence *of v, where*

$$\Delta_v = (\delta_v^0, \delta_v^1, \ldots, \delta_v^L).$$

The notation δ_v^r is a generalization of the number of *adjacent* symbols at distance r from a sequence v. To compute the degree sequence of an unconstrained codeword is trivial. For any unconstrained codeword, the r-th degree is the binomial coefficient, *i.e.*,

$$\delta_v^r = \binom{n}{r}, \quad r = 0, 1, \ldots, n, \text{ independent of } v \in U.$$

In general no systematic result is available when the set consists of constrained sequences. For the d-constraint case, since the last d-bits are assumed to be all zeros,

$$\Delta_v = (\delta_v^0, \delta_v^1, \ldots, \delta_v^{l-d}), \quad v \in C.$$

Definition 2.3 (n-sequence) *A binary sequence $v \in C$ is called an n-sequence if*

$$\sum_{r=j}^{l-d} \delta_v^r \geq \sum_{r=j}^{n} \binom{n}{r}, \quad j = 0, 1, \ldots, n.$$

In short, a sequence is an n-sequence if it has a minimum number of neighboring sequences at a far enough Hamming distance.

The third necessary condition can be stated as:

NC3. Degree sequence test

For a sequence $v \in C$ to be a member of a Hamming map, it is necessary that v be an n-sequence.

When NC3 is applied to each sequence as in [19], it requires $N_d(l)$ computations. To reduce this computation, the set C is partitioned into smaller sets C_i as:

$$C_0, C_1, \ldots, C_m,$$

where $m = \lfloor \frac{l}{d+1} \rfloor$ and $v \in C_i$ if and only if the Hamming map weight of v, $w_H(v) = i$. The symbol $\lfloor x \rfloor$ denotes the floor function of x, *i.e.*, the greatest integer not exceeding x. If each $v \in C_i$ is an n-sequence, set C_i is called an n-set. Note every sequence in an n-set passes NC3. The following theorem, the proof of which is given in Appendix A, results in a reduced amount of computation.

Theorem 2.1 (n-set) *If C_i is an n-set for some $i \geq 0$, then so is C_{i+1}.*

To exploit this property, the third test can be restated as follows.

NC3$'$. Refined degree sequence test

Apply NC3 to the partitioned sets in the order C_0, C_1, ... until an n-set C_i

is reached for some i (hopefully small enough). Then all other sets $C_j, j \geq i$ will be n-sets by Theorem 2.1.

To show the improvement, Table 2.3 tabulates the partitioned sets for the case of $d = 1$ and $l = 7$. The set C_1, having 6 out of 21 sequences, turned out to be an n-set and thus the test is not required for the rest of the sets C_2, C_3, and C_4—a saving of about 67% . Table 2.4 lists the actual number of sequences tested for various values of l when $d = 1$.

Table 2.3: Partitioned sets and degree sequence when $d = 1$ and $l = 7$.

C_i	$v : \Delta_v$	0	1	2	3	4	5	6
C_0	0000000	1	6	10	4	0	0	0
C_1	0000010	1	5	8	6	1	0	0
	0000100	1	4	6	7	3	0	0
n-set	0001000	1	4	7	7	2	0	0
	0010000	1	4	7	7	2	0	0
	0100000	1	4	6	7	3	0	0
	1000000	1	5	8	6	1	0	0
C_2	0001010	1	4	6	5	4	1	0
	0010010	1	3	6	7	4	0	0
	0100010	1	3	6	6	4	1	0
n-set	1000010	1	4	7	6	3	0	0
	0010100	1	3	5	5	5	2	0
	0100100	1	2	5	6	5	2	0
	1000100	1	3	6	6	4	1	0
	0101000	1	3	5	5	5	2	0
	1001000	1	3	6	7	4	0	0
	1010000	1	4	6	5	4	1	0
C_3	0101010	1	3	4	5	4	3	1
	1001010	1	3	5	6	4	2	0
n-set	1010010	1	3	5	6	4	2	0
	1010100	1	3	4	5	4	3	1

Table 2.4: Number of sequences tested for NC3 when $d = 1$.

n	l	n-set	$NC3$	$NC3'$
2	4	C_0	5	1
3	5	C_1	8	5
3	6	C_0	13	10
4	7	C_1	21	7
5	8	C_2	34	22
5	9	C_1	55	9
6	10	C_2	89	38
7	11	C_2	144	113
8	12	C_2	233	57
8	13	C_3	377	188
9	14	C_4	610	455

2.3 Graph Search Technique

The tree search algorithm is useful to find a single Hamming map if it exists. An efficient technique which employs a graph theoretic algorithm to find all Hamming maps is now introduced. The motivation for this graph search arises from the following observation. In the linear distance matrix D_u, the largest entries n appear along the main cross diagonal, *i.e.*,

$$d^u_{i\hat{i}} = d(u_i, u_{\hat{i}}) = n, \; \hat{i} = q - 1 - i, \; i = 0, 1, \ldots, q - 1$$

This pair of indices $(i, \hat{i})$ is called a *matched slot pair.* In the matched slot are the complementary pair of codewords $(u_i, u_{\hat{i}})$, where $u_{\hat{i}}$ is obtained from u_i by negating each bit.

At this point, a question can be raised. Instead of choosing one codeword at a time as in the tree search algorithm, is it more efficient to choose a set of q codewords as a whole? The investigation of this problem is a major part of the research. The answer entails the determination of efficient ways to:

1. Choose a set M;

2. Evaluate the set M.

As was pointed out, choosing such a set arbitrarily involves $\begin{pmatrix} N_d(l) \\ q \end{pmatrix}$ combinations. This difficulty is alleviated by exploiting a graph theoretic problem known as **maximum matching** to select a set of q constrained sequences. To avoid the enormous computation of $q!$ permutations needed to test the set, an efficient method termed **canonical ordering algorithm** is developed in a subsequent chapter. First, simple examples are presented to illustrate the idea of maximum matching and its relationship to the problem of determining a Hamming map.

2.3.1 Examples

Let G_u denote the *weighted* complete graph of the unconstrained set U where: the vertices represent the unconstrained codewords of U; every pair of distinct vertices (u, v) is connected by an edge weighted with the Hamming distance $d(u, v)$ between the two unconstrained codewords u and v. The weighted complete graph G_c of the constrained set C is defined similarly.

The mapping between U and V now becomes a mapping between G_u and a *subgraph* G_v of G_c. Fig. 2.4, for example, shows the weighted complete graphs G_u and G_c when $d = 1$, $n = 2$, $l = 4$, and $N_d(l) = 5$. Noting that any edge weight is at least unity, one can eliminate all the unit-weight edges from graphs without losing information. This elimination, however, greatly facilitates the task of locating a subgraph G_v. This can be seen by realizing that after the elimination of unit-weight edges, the edge reduced graph—two solid edges e_1 and e_2—is simply a *maximum matching* of G_u.

Maximum matching of a graph is an well-known subject in graph theory. Extensive discussion on this subject can be found in many texts such as Bondy and Murty [10]. For completeness, the applicable graph theoretic

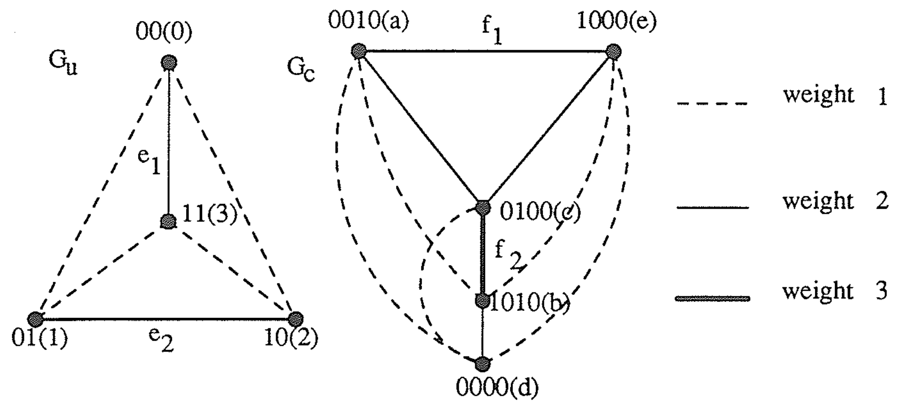


Figure 2.4: The weighted complete graphs G_u and G_c: $d = 1, n = 2, l = 4$, and $N_d(l) = 5$.

terminology is summarized.

A *matching* M of a graph G is a set of edges such that no two are adjacent in G. The two vertices of an edge in M are said to be *matched.* A matching M *saturates* a vertex v, and v is said to be M*-saturated*, if some edge of M is incident with v. If every vertex of G is M-saturated, the matching is *perfect.* M is a *maximum matching* if G has no matching M' with $\mid M' \mid > \mid M \mid$. To determine the *size* of a matching, either the edges or vertices in the matching are counted.

Returning to the example, note that the desired Hamming mapping may be obtained from a maximum matching M_c of the edge reduced graph G_c. By inspection, one can easily verify that there are four different maximum matchings for this goal, from M_1 through M_4, as illustrated in Fig. 2.5. Consider the matching $M_1 = \{f_1, f_2\} = \{(a, e), (c, b)\}$ as an example. Note

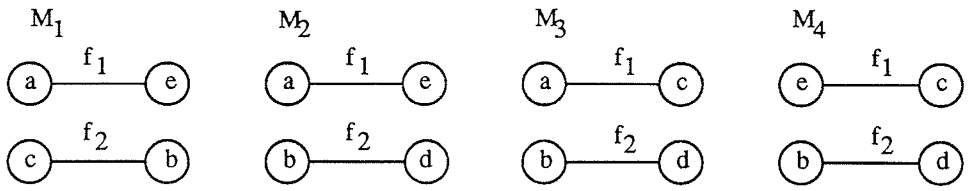


Figure 2.5: Maximum matchings of constrained graph G_c with edges of weight two or more: M_1–M_4.

that the weights of edges e_1, e_2, f_1, and f_2 are two or more. The mapping $f' : M_u = \{e_1, e_2\} \rightarrow M_1 = \{f_1, f_2\}$ described by

$$\begin{aligned} f_1 &= (a, e) = f'(e_1) = f'(0, 3) \\ f_2 &= (c, b) = f'(e_2) = f'(1, 2) \end{aligned} \tag{2.9}$$

determines a Hamming map we are looking for. The Hamming map $f : U \rightarrow V$ induced by the above map f' becomes:

$$f : (0, 3, 1, 2) \ \rightarrow \ (a, e, c, b). \tag{2.10}$$

That is, replacing convolutional codewords $(00, 01, 10, 11)$ with constrained codewords $(0010, 0100, 1010, 1000)$ in that order preserves the free distance.

It is interesting to note that the edges of a matching can be reordered—flipping and interchanging edges—to produce many other Hamming maps. This is best illustrated by the example shown in Fig. 2.6, where the matching

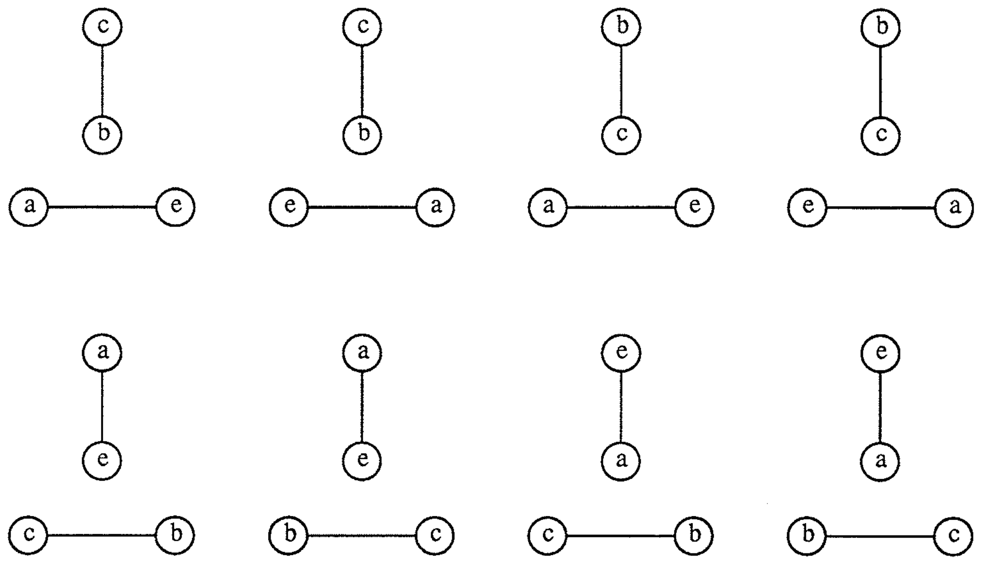


Figure 2.6: Eight mappings produced from the matching M_1.

M_1 produces 8 different Hamming maps. In this way, a total of 32 (4×8) different mappings can be obtained which corresponds to an exhaustive search. This property is generalized later with the introduction of **conjugate maps**.

For an arbitrary codeword length n, the graph map search becomes nontrivial because the reduced graph G_u contains edges of different weights $2, \ldots, n$. This difficulty can be seen by taking the $n = 3$ case, as illustrated in Fig. 2.7. The reduced graph G_u now consists of two copies of K_4 (a 4-vertex complete graph) with edges of weight two, and a perfect matching M_u which spans the two K_4's by the use of four edges of weight three.

In the constrained counterpart, it is not obvious how to locate these components properly. Nonetheless, it turns out that no perfect matching of the constrained graph, which uses edges of weight three or more, exists. This means that there is no Hamming mapping from a rate 2/3 (k/n) convolutional code to a rate 2/5 (k/l) constrained code. When this happens, suffix and/or prefix construction [19] could be used. That is, adding 00 or 10 to either one of the two copies of the Hamming map for length $l - 1$ creates a rate $k/(l + 1)$ trellis code. In Chapter 3, the two problems—choosing and evaluating the maximum matchings—are investigated in detail.

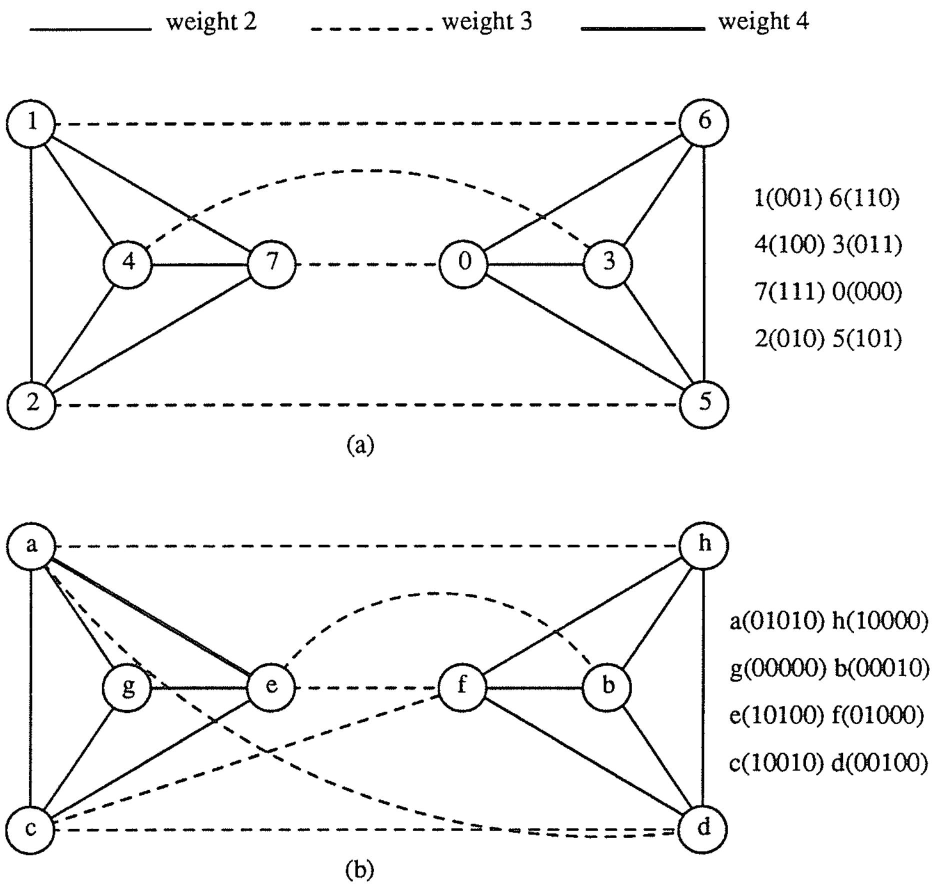


Figure 2.7: The weighted complete graphs G_u and G_c: $d = 1, n = 3, l = 5$, and $N_d(l) = 8$.

Chapter 3

Canonical Ordering Algorithm

The problem of determining Hamming maps, as shown in Chapter 2, can be cast as that of examining each matching. This chapter describes a *depth-first search* (DFS) algorithm which lists each maximum matching, and a *canonical ordering* algorithm which evaluates the maximum matching. The canonical ordering algorithm produces a Hamming map through a rearrangment of the matching edges. Prior to *canonical ordering*, the distance matrix associated with codewords represented by the vertices of the maximum matching is *balanced* so that it is both symmetric and cross-symmetric. This process of balancing reduces the computation required for canonical ordering.

3.1 Graph Definition

The various graphs discussed in this thesis are characterized in graph theoretic terms. A weighted graph G is a triple $(V(G), E(G), W(E))$ consisting of a set $V(G)$ of vertices, a set $E(G)$ of edges, and a function W from E into Z^+, the positive integers. An edge e of G is an unordered pair of joined

Table 3.1: Graph parameters.

Graph	$V(G)$	$\nu(G)$	$E(G)$	$\varepsilon(G)$
G_u	$\{u_0, \ldots, u_{q-1}\}\ u_i \in U$	$q = 2^n$	$(u_i u_j)\ \forall i, j$	$\binom{q}{2}$
G_c	$\{c_0, \ldots, c_{r-1}\}\ c_i \in C$	$r = N_d(l)$	$(c_i c_j)\ \forall i, j$	$\binom{r}{2}$
G_v	$\{v_0, \ldots, v_{p-1}\}\ v_i \in C$	$p \leq N_d(l)$	$W(e) \geq n$	$\ll \binom{r}{2}$

vertices u and v, *i.e.*, $e = uv = vu$. For an edge e, $W(e)$ is called the weight of e. Symbols $\nu(G)$ and $\varepsilon(G)$ denote the number of vertices and edges in graph G. Using these symbols, the complete weighted graphs G_u and G_c introduced in Chapter 2 are parameterized in Table 3.1. Note that $\nu(G_v)$ is the number p of d-constrained sequences which have passed the necessary conditions NC2 and NC3. Because $e \in E(G_v)$ if and only if $W(e) \geq n$, the subgraph G_v will have considerably fewer edges than the complete graph G_c, reducing the search effort consequently. When a maximum matching M of G_v is found, each edge of M, a pair of d-constrained sequences at distance n or greater, will replace an edge of weight n in G_u (See Fig. 2.7 in Chapter 2 for example).

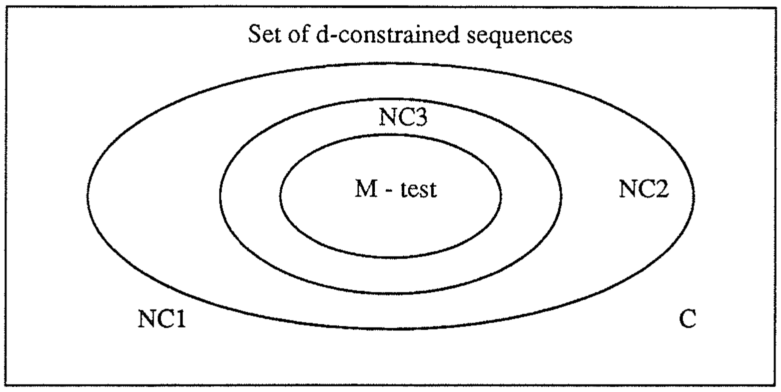


Figure 3.1: Conjectured relationship between tests.

The Hamming map algorithm consists of two steps:

Step 1. ***M*-test** Find a maximum matching M of G_v using a known algorithm. If $|M| < q = 2^n$ then stop.

Step 2. Canonical Arrangement Evaluate each matching M of G_v.

It is interesting to note that the M-test can be used as a necessary condition for the existence of a Hamming map. It is suspected that this M-test is stronger than NC3. That is, it is conjectured that failure of NC3 implies failure of M-test, whereas failure of the M-test may or may not imply failure of NC3, as illustrated in Fig. 3.1. No counter example has been found to disprove this conjecture. Table 3.2 and 3.3 support this claim by showing test results for various conditions. The strength of the M-test is demonstrated by verifying the nonexistence of a Hamming map for rates 7/13 and 8/14 when $d = 1$, and rates $2/7, 5/13$, and 6/15 when $d = 2$.

The *Canonical arrangement* step consists of two algorithms which respec-

Table 3.2: Test results from NC3 and M-tests when $d = 1$: Y–Hamming map may exist, N–Hamming map does not exist.

k	n	l	2^n	$N_d(l)$	r_u	r_c	NC3	M-test
1	2	4	4	5	1/2	1/4	5(Y)	4(Y)
2	3	5	8	8	2/3	2/5	7(N)	6(N)
2	3	6	8	13	2/3	2/6	13(Y)	12(Y)
3	4	7	16	21	3/4	3/7	20(Y)	18(Y)
4	5	8	32	34	4/5	4/8	29(N)	22(N)
4	5	9	32	55	4/5	4/9	54(Y)	48(Y)
5	6	10	64	89	5/6	5/10	83(Y)	66(Y)
6	7	11	128	144	6/7	6/11	123(N)	82(N)
6	7	12	128	233	6/7	6/12	226(Y)	184(Y)
7	8	13	256	377	7/8	7/13	349(Y)	252(N)*
8	9	14	512	610	8/9	8/14	525(Y)	310(N)*

Table 3.3: Test results from NC3 and M-tests when $d = 2$: Y–Hamming map may exist, N–Hamming map does not exist.

k	n	l	2^n	$N_d(l)$	r_u	r_c	NC3	M-test
1	2	6	4	6	1/2	1/6	6(Y)	6(Y)
2	3	7	8	9	2/3	2/7	8(Y)	6(N)*
2	3	8	8	13	2/3	2/8	12(Y)	12(Y)
3	4	9	16	19	3/4	3/9	15(N)	12(N)
3	4	10	16	28	3/4	3/10	27(Y)	22(Y)
4	5	11	32	41	4/5	4/11	31(N)	20(N)
4	5	12	32	60	4/5	4/12	55(Y)	42(Y)
5	6	13	64	88	5/6	5/13	70(Y)	44(N)*
5	6	14	64	129	5/6	5/14	116(Y)	86(Y)
6	7	15	128	189	6/7	6/15	141(Y)	72(N)*

tively list and evaluate the matchings M. The evaluation algorithm takes a matching M and examines each submatching H of size q for a possible canonical arrangement. The algorithm that lists the matchings has been developed using a depth-first search as described in the following section.

3.2 Listing the maximum matchings

A recursive algorithm based on a depth first search has been developed to list all maximum matchings of the constrained graph G. Given a maximum matching, a partition on the set of all maximum matchings into two disjoint sets is made once per each matching edge e: a set of maximum matchings M_e which contain edge e and a set of maximum matchings M'_e which do not contain edge e. After each partition, the problem of listing all maximum matchings of m edges is divided into two smaller problems: listing all maximum matchings of $(m-1)$ edges from a vertex reduced graph and that of m edges from an edge reduced graph. Terminology and some fundamental results for the maximum matching problem are described first.

3.2.1 Preliminary

As introduced in section 2.3, a matching M of a graph G is a set of edges with no common endpoints. An **M-alternating path** P is a path whose edges are alternately *in* M and *not in* M. See Figure 3.2. Note that the term *alternating* is relative to a specific matching M. When M is empty, for

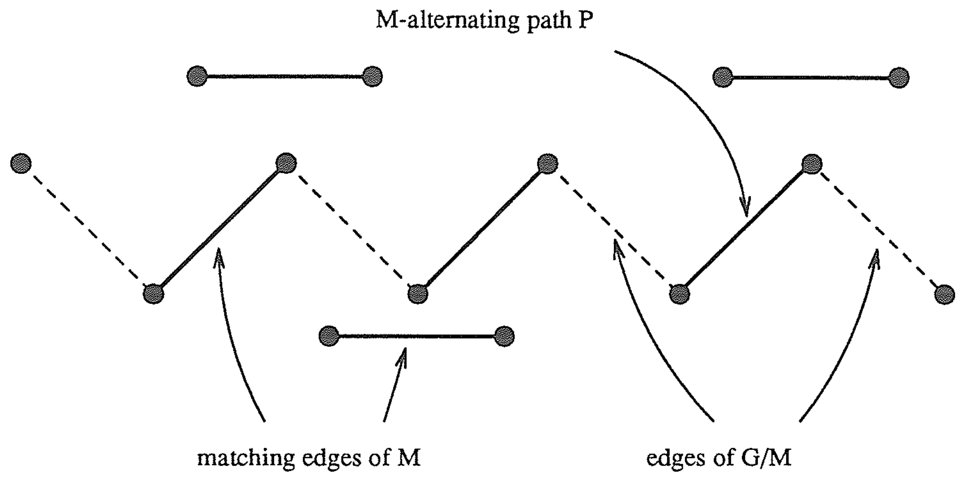


Figure 3.2: An M-alternating path.

instance, then any single edge is an M-alternating path. An alternating path P with both endpoints unsaturated (unmatched) is called an augmenting path relative to M, or an **M-augmenting path**.

Given an augmenting path P, a bigger matching M' can be obtained by an *exclusive-OR* operation on the edges, $M' = M \oplus P$. The exclusive-OR operation is known as an **augmenting** operation. The new matching M' consists of those edges that are in M or P, but not in both. In this way M' has one more edge than M, *i.e.*, $|M'| = |M| + 1$. Figure 3.3 illustrates a bigger matching $M' = \{(1,8),(2,6),(3,7),(4,9)\}$ obtained by the *augmenting* operation.

The determination of a maximum matching is greatly facilitated by a theorem, proven by Berge([7],1957): *A matching M in G is a maximum matching if and only if G contains no M-augmenting paths.*

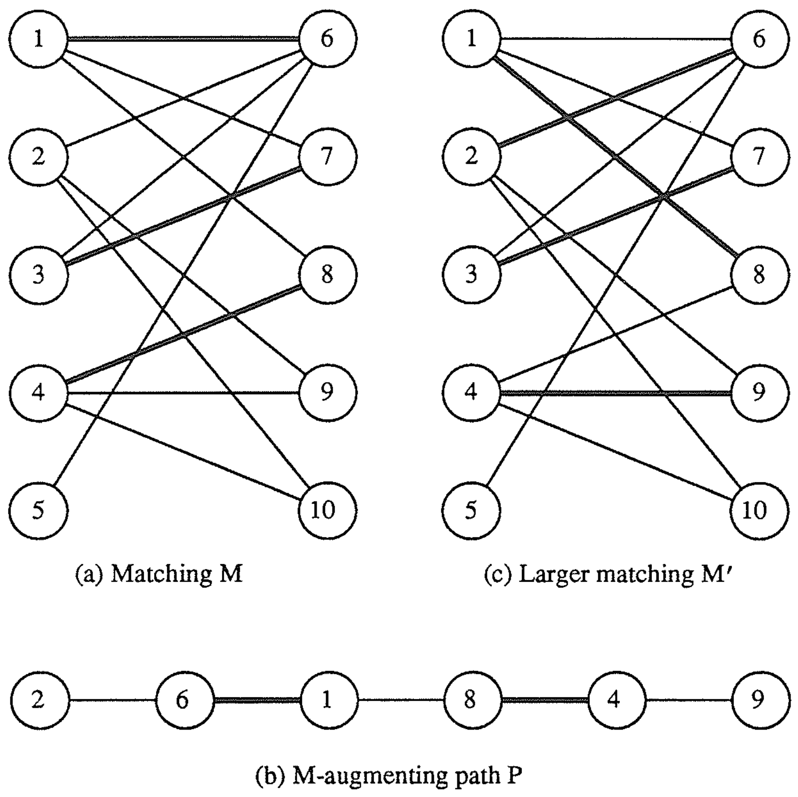


Figure 3.3: A graph and an M-augmenting path.

The search for an M-augmenting path with origin u involves *growing* an M-alternating tree H rooted at u. A tree T is a connected graph that contains no cycle. A subtree $H \subseteq G$ is called an M-alternating tree H rooted at u if (1) $u \in V(H)$; (2) $\forall v \in H$, the unique (u, v)-path in H is an M-alternating path. A maximum matching algorithm follows the alternating paths as outlined below.

Step 1. Start with an empty matching $M = \phi$.

Step 2. Find an M-augmenting path P and replace M by $M \oplus P$.

Step 3. Repeat **step 2** until no further augmenting paths exists, at which point M is a maximum matching.

For the the purpose of Hamming maps, two maximum matching algorithms are used in this thesis: The Hungarian algorithm [10] and Edmonds algorithm [15]. The Hungarian algorithm can find a maximum matching in bipartite graphs. A *bipartite graph* is one whose vertex set can be partitioned into two subsets X and Y, so that each edge has one end in X and the other end in Y. Edmonds algorithm can find a maximum matching in nonbipartite graphs. Both are based on a **breadth-first search**.

In a *breadth-first search*, vertices are visited in order of increasing distance d from the starting point v, where d is simply the number of edges in a shortest path. Beginning with $d = 0$, the *breadth-first search* considers in turn each vertex x of distance d from v. This is repeated until no new vertices are found. By examining all edges incident with x, all vertices of distance $d+1$ from v are processed.

On the other hand, the **depth-first search** is a generalization of a *preorder* traversal of a tree as illustrated in Fig. 3.4 [5]. The starting vertex may be determined by the problem or may be chosen arbitrarily. When each new vertex v is visited, a path is followed as far as possible, *visiting* or processing all the vertices along the way, until a *dead end* is reached. A *dead end* is a vertex all of whose neighbors (vertices adjacent to it) have been visited already. At a dead end the search backs up along the last edge traversed and branches out in another direction. This has the effect of visiting all vertices in one subgraph adjacent to v, say G_1, before going to a new one.

To describe the various algorithms, control structures similar to Pascal and C languages are used. In the following algorithm, the instruction *mark x visited* is used when the desired processing of the vertex x is completed.

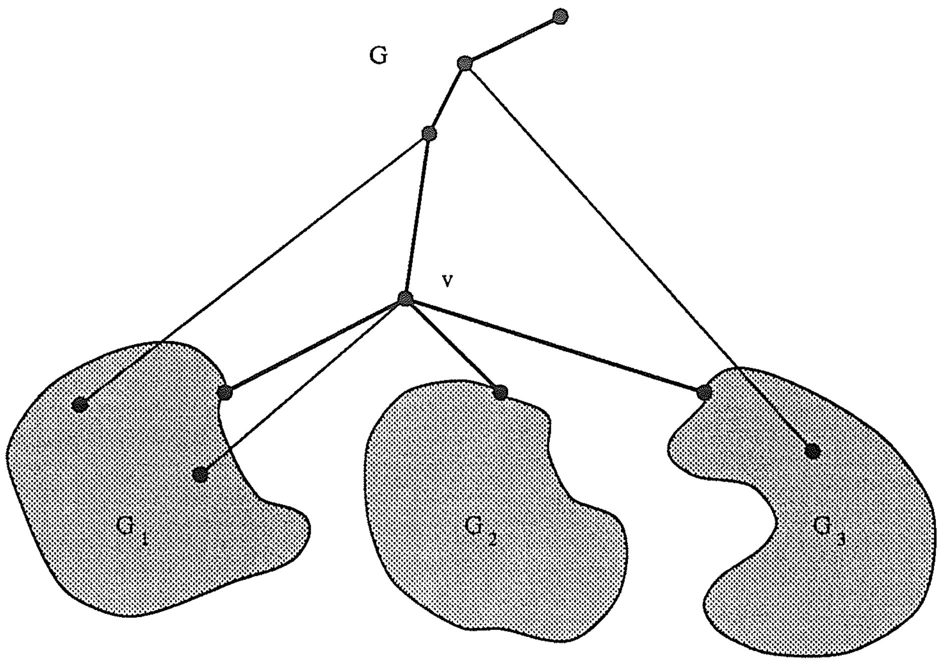


Figure 3.4: Depth-first search.

The *breadth-first search* uses a *queue* Q to store and process vertices in the order of arrival.

Algorithm Breadth-First Search

```
Procedure BFS(v: VertexType);
  {
    var
      Q : Queue;
      x, w : VertexType;
    Q := ϕ; {begin with an empty queue}
    place v in Q and mark v visited;
    do {
      select x from Q;
      for each unmarked vertex w adjacent to x {
        place w in Q and mark w visited;
      } {end for}
    } while Q is not empty; {until all of Q has been processed}
  } {end BFS}
```

Since a depth-first search does not back up from a vertex x until every edge from x has been examined, it has a very simple recursive description.

Algorithm Depth-First Search

```
Procedure DFS(v: VertexType);
  {
    var
      w : VertexType;
    mark v visited;
    while there is an unmarked vertex w adjacent to v {
      DFS(w);
    } {end while}
  } {end DFS}
```

3.2.2 Listing the maximum matchings

Consider an edge $e = xy$ of G as depicted in Fig. 3.5. A maximum matching M of G may or may not contain the edge e. In an *edge reduced graph* $G - e$, no maximum matching can contain the edge e. The end vertices x and y, however, can be contained in some maximum matchings of the graph $G - e$. On the other hand, in the *vertex reduced graph* $G - \{x, y\}$, no maximum matching (perhaps smaller) can contain x and y. Fig. 3.6 shows graphs reduced by an edge or its end vertices.

Based on this observation, the strategy for listing maximum matchings is to partition the set of all maximum matchings according to whether a maximum matching uses a particular edge. Suppose that a maximum matching M called a **base matching** has m edges ($2m$ constrained sequences), *i.e.*,

$$M = \{e_0, e_1, \ldots, e_{m-1}\}, \; e_i = (v_i, \hat{v}_i) \tag{3.1}$$

When $|M| = m$, it will be called an m-matching M to save notation. The set of all m-matchings $\{M\}$ of G is partitioned into two mutually exclusive groups:

$$\{M\} = \{M_0\} \cup \{M_0'\} \tag{3.2}$$

The first group $\{M_0\}$ contains matchings which use the edge e_0 and the second group $\{M_0'\}$ contains matchings which do not use the edge e_0. In other words, a matching M is in $\{M_0\}$ if $e_0 \in M$, otherwise, it is in $\{M_0'\}$. This hierarchy is depicted in Fig. 3.7 as a *partition tree.*

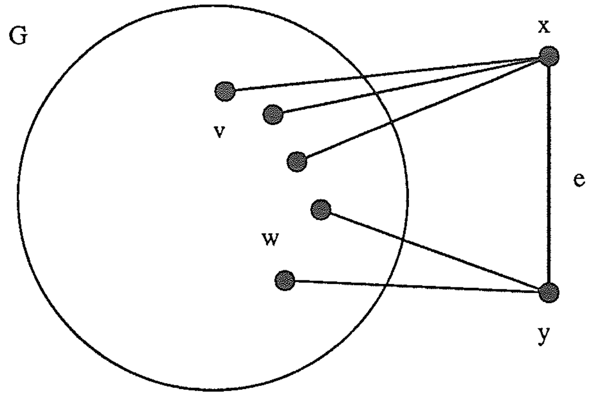


Figure 3.5: A graph with edge $e = xy$.

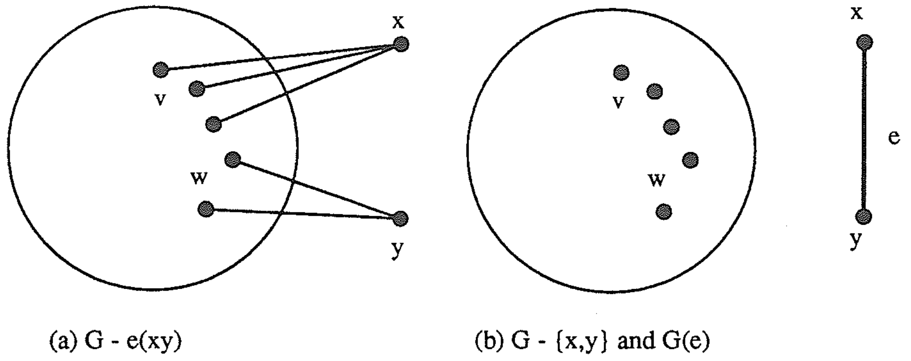


Figure 3.6: An edge or vertex reduced graph.

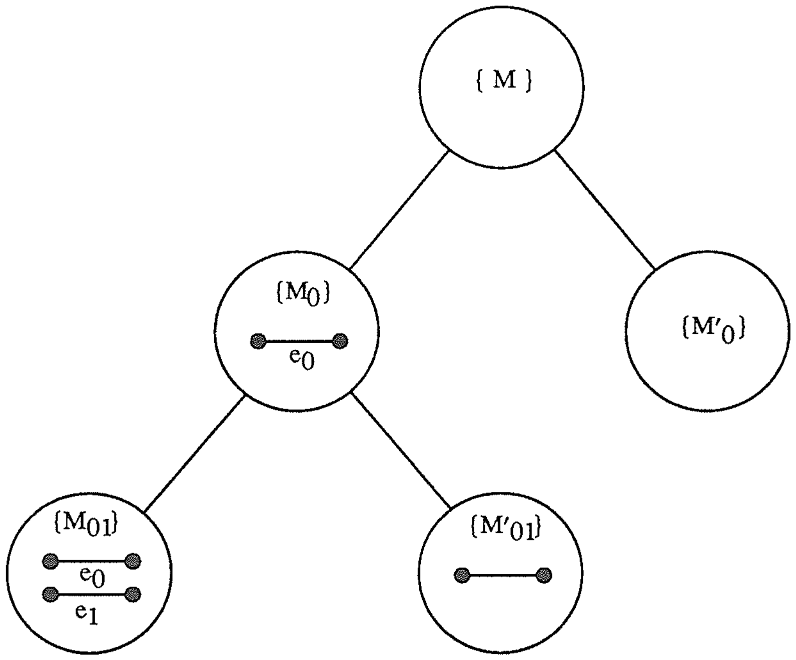


Figure 3.7: A partition tree of the set of maximum matchings.

Consider the right branch of the partition tree first, *i.e.*, the set $\{M_0'\}$ of matchings that do not use edge e_0. Each of these m-matchings is in the edge reduced graph, $G_A := G - e_0$. Let M_A denote a part of m-matching M as $M_A := M - e_0$. Note that the two end vertices x_0 and y_0 are still in the reduced graph G_A, albeit they have become unsaturated. Therefore the reduced matching M_A is no longer a maximum or m-matching in either G or G_A. The determination of all m-matchings in G_A involves growing M_A-alternating trees rooted at one of the end vertices x_0 or y_0, and is described later as a depth-first search.

Now consider the left branch of the partition tree, the set of maximum matchings that do use edge e_0. Let a two-tuple (G_B, M_B) denote the graph with a base maximum matching obtained from M by the removal of the end vertices x_0 and y_0:

$$\begin{aligned} G_B &:= G - \{x_0, y_0\} \\ M_B &:= M - e_0 = \{e_1, e_2, \ldots, e_{m-1}\} \end{aligned} \tag{3.3}$$

It should be emphasized that the $(m-1)$-matching M_B is a **maximum matching** of this vertex reduced graph G_B. This can be seen by realizing that G_B has lost two vertices which were previously M-saturated in G. Consequently, the size of a maximum matching M_B of G_B is one less than that of M. The m-matchings in the left branch of the partition tree are obtained as the union of $\{e_0\}$ and all $(m-1)$-matchings of this vertex reduced graph

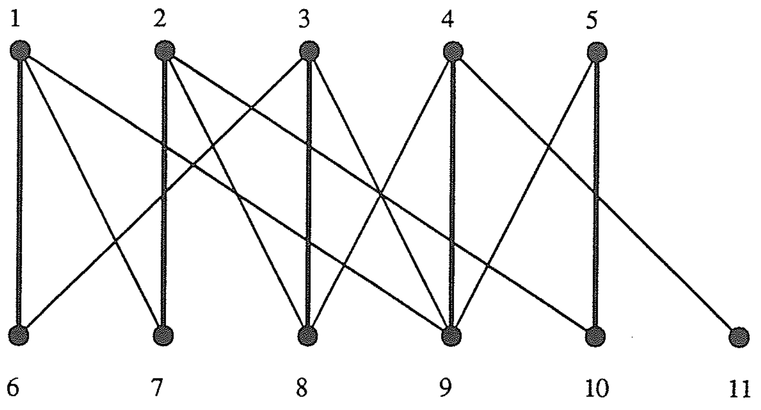
G_B.

$$\{M_0\} = \{M_B\} + \{e_0\} \tag{3.4}$$

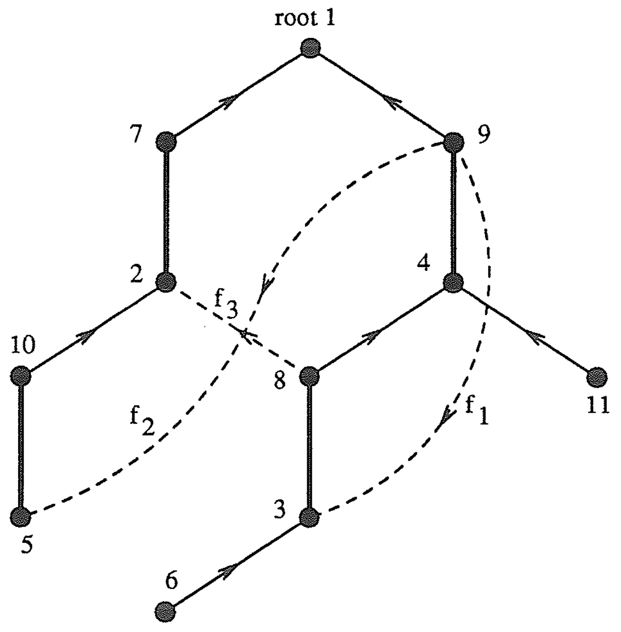
The task of listing $(m-1)$-matchings $\{M_B\}$ of the vertex reduced graph G_B is the same as that of listing m-matchings $\{M\}$ of the original graph G. Therefore with the $(m-1)$-matching M_B as the base matching of graph G_B, a new partitioning of the set of $(m-1)$-matchings $\{M_0\}$ into $\{M_{01}\}$ and $\{M_{01'}\}$ is made to move further down along the partition tree based on the edge e_1. In the same way, a matching in $\{M_{01}\}$ uses both edges e_0 and e_1 and a matching in $\{M_{01'}\}$ uses e_0 but not e_1.

This partitioning is repeated for the left branch until all the edges in the base m-matching M have been examined, each time producing two disjoint sets of matchings. At depth d of the partition tree, the left branch denotes the set of all m-matchings which contains the base edges $\{e_0, e_1, \ldots, e_{d-1}\}$, and the right side branch $\{e_0, e_1, \ldots, e_{d-2}\}$ but not e_{d-1}. In this way, all m-matchings of the graph G can be listed without repetition.

To explain the DFS-algorithm for the new m-matchings of $G_A = G - e_0$, consider the bipartite graph G of Fig. 3.8. The graph has 11 vertices with a 5-edge matching $M = \{(1,6), (2,7), (3,8), (4,9), (5,10)\}$ as a base. The edge reduced graph $G - e(1,6)$ in Fig. 3.8(b) is redrawn as an M_A-alternating tree rooted at vertex 1, one end of $e = (1,6)$. Note that the link edges f_1, f_2, and f_3 are not part of the tree. Two M_A-augmenting paths P—(11,4,9,1) and (6,3,8,4,9,1)—can bring about new m-matchings of G.



(a) A bipartite graph G



(b) BFS tree of G - e(16)

Figure 3.8: A bipartite graph and an M_A-alternating tree.

The M_A-alternating tree of Fig. 3.8(b) was based on a *breadth-first search* tree. The BFS tree structure depends on the initial implementation of a graph, represented as an adjacency linked list. For example, if the link edge f_3 were encountered earlier than the edge (2,10), the path (6,3,8,2,7,1) would be an M_A-augmenting path. In this respect, BFS trees are useful when an exhaustive search for augmenting paths is not required.

On the other hand, DFS trees extend as far as possible until an M_A-augmenting path from a vertex v is found. After the new matching from this augmenting path is evaluated, the vertex v is *marked as unvisited* (removed from the tree). Also when each adjacent vertex v to *root* vertex u is examined, u is marked as unvisited because it might be in another alternating path, perhaps from some other branches of the tree.

Now consider the M_A-augmenting path $P = (11, 4, 9, 1)$ again. The m-matching M obtained by the augmenting operation can be expressed as a union of two sets of matching edges:

$$\begin{aligned} M &= M_A \oplus P = M(P) + M(G_A/P) \\ &= \{(11,4),(9,1)\} + \{(2,7),(3,8),(5,10)\} \end{aligned} \tag{3.5}$$

The two matching edges of $M(P)$ are the direct result of the augmenting operation. The three edges of $M(G_A/P)$, however, remained unchanged by the augmenting operation. This situation illustrates the following important consideration:

There may be other m-matchings which could be missed if the

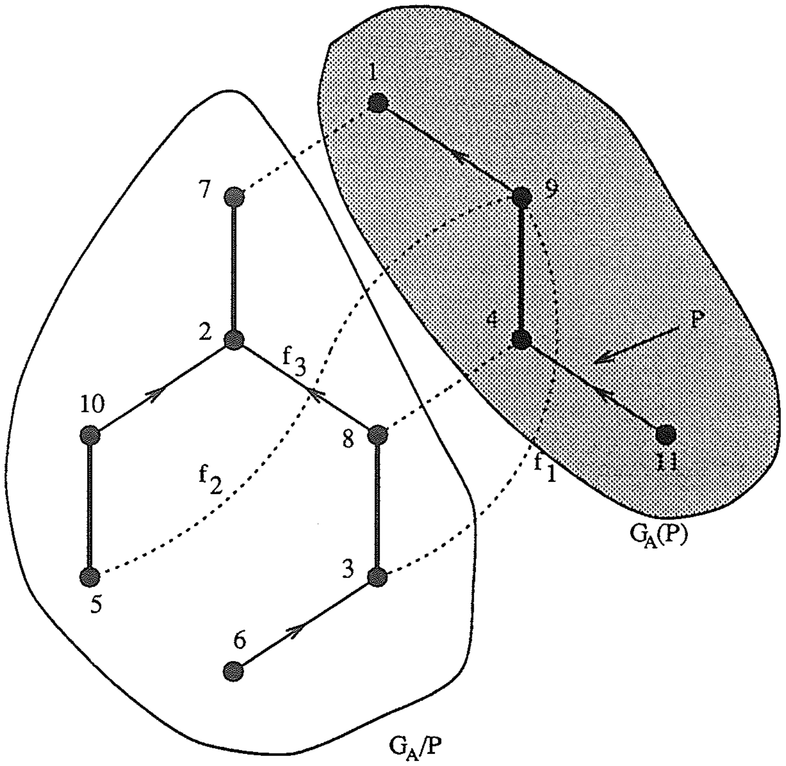


Figure 3.9: Temporary removal of P for possible children matchings.

search backs up immediately from the M_A-augmenting vertex, say 11.

A close examination indeed reveals many such hidden matchings which are termed **child matching**. Fig. 3.9, 3.10 illustrates two child matchings, for example.

$$M(P_1) = \{(11,4),(9,1)\} + \{(2,8),(3,6),(5,10)\}$$

$$M(P_2) = \{(11,4),(9,1)\} + \{(2,7),(3,6),(5,10)\}$$

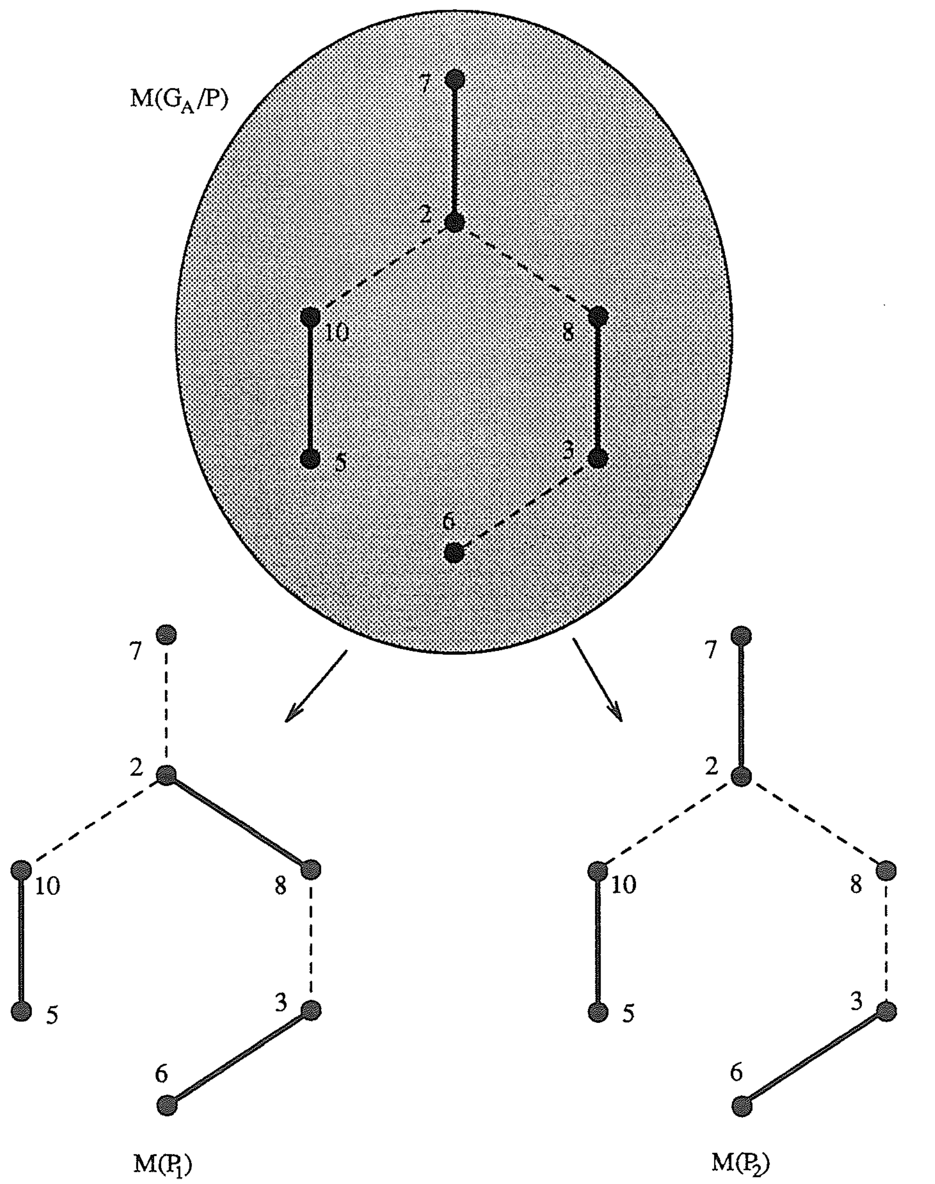


Figure 3.10: Child matchings.

To find the *child* m-matchings, the portion of the M_A-augmenting path P is removed temporarily. The rest of the graph G_A/P will have a maximum matching of size at most $|M| - |P| = m - p$ edges. The listing of $(m - p)$-matchings of the two-tuple $(G_A/P, M(G_A/P))$ again becomes the same problem as that of m-matchings of $(G, M(G))$, the original graph.

The following description summarizes the listing algorithm in three procedures: *Main*, *DFS_Match*, and *Process_Match*.

The procedure *Main* merely states the strategy described before: *partitioning* and *building M_A-alternating trees*, once for each matching edge. To begin the enumeration, Edmonds algorithm is used to find a base m-matching M from G.

The procedure *DFS_Match* invokes the procedure *Process_Match*, when a new M_A-augmenting path (thus a new m-matching) is found. The procedure *Process_Match* first *evaluates* the new m-matching M in an attempt to produce a Hamming map(s) from the set of constrained sequences represented by the matching edges. This *evaluation* procedure will be described later in a separate section. It then begins to look for child $(m - p)$-matchings. Each child matching, if it exists, is added to $M(P)$, yielding an m-matching of G_A. Note that a child $(m - p)$-matching of G_A/P may exist only when $|M| > |P|$. That is, if all of $2m$ vertices of the new matching are in the augmenting path, there cannnot be a separate matching edge which is not in a *maximum matching.*

Algorithm 1: Main

```
Procedure Main;
  {
    Var
      $G = (V, E)$: GraphType;
    find a maximum matching $M$; {by Edmonds algorithm}
      $M := \{e_0, e_1, \ldots, e_{m-1}\}, e_i := (x_i, y_j)$
    for each edge $xy$ of $M$ {
      remove edge $xy$; $\{G_A := G - e(xy)\}$
      for each unsaturated vertex $u$ do DFS_Match($u$);
      delete vertices $x$ and $y$; $\{G_B := G - \{x, y\}\}$
    } {end for}
  } {end Main}
```

Algorithm 2: All M_A-augmenting paths (recursive)

```
Procedure DFS_Match(u: VertexType);
  {
    var
      PrevPt: array of VertexType;
      M: Matching;
      v, w : VertexType;
      mark u visited;

    for each unmarked vertex v adjacent to u {
      PrevPt[v] = u; {representing P}
      if v is matched to w {
        mark v visited;
        DFS_Match(w);
        mark v unvisited;
      } {end for}
      else {
        Augment(v); {an $M_A$-augmenting path from v}
        {we have a new maximum matching M}
        Process_Match(M);
        mark v unvisited {backs up };
      } {end else}
    } {end for}
    mark u unvisited;

  } {end DFS_Match}
```

Algorithm 3: Process matching

```
Procedure Process_Match(M: Maximum Matching);
  {
    first, evaluate matching $M$; {Canonical Ordering}
    $P$ := augmenting path;
    if $|P| < |M|$ { {take care of child matchings};
      $G := G - P$; {temporary removal of the augmenting path}
      $M := M - M(P)$;
      for each edge $xy$ of $M$ {
        remove edge $xy$; $\{G := G - e(xy)\}$
        for each unsaturated vertex $u$ do DFS_Match($u$);
        delete vertices $x$ and $y$; $\{G := G - \{x, y\}\}$
      } {end for}
      $M := M + M(P)$; {restore matching}
      $G := G + P$;{restore graph}
    } {end if}
  } {end Process}
```

3.3 Canonical Ordering

In the previous section, a systematic way of listing the maximum matchings of a constrained distance graph has been developed. To determine the Hamming map, the set of q codewords ($q/2$ edges) of an m-matching M has to be ordered properly. An exhaustive trial would require $q!$ examinations for each submatching—$8! = 40320$ when $k/n = 2/3$, $16! \approx 2.09 \times 10^{13}$ when $k/n = 3/4$, etc.

In the following, an efficient algorithm termed *canonical ordering* which determines a Hamming map is described. Note that when $k/n = 1/2$, no additional ordering is required since then any subset of 4 codewords represented by 2 matching edges in any order produces a Hamming map.

3.3.1 Matrix balancing

The key idea is to make a constrained Hamming distance matrix cross-symmetric by *balancing* the distances between pairs of matched codewords. This balancing greatly facilitates the proper arrangement of matched codewords into an Hamming map when one exists. It is convenient to define two kinds of matrix symmetries.

Definition 3.1 (symmetry) *A $q \times q$ matrix A is called* symmetric *if $a_{ij} = a_{ji}$, $\forall i, j$, and* cross-symmetric *if $a_{ij} = a_{\hat{j}\hat{i}}$, where pairs $(i, \hat{i})$ and $(j, \hat{j})$ are matched slot pairs, that is, $\hat{i} = q - 1 - i$, and $\hat{j} = q - 1 - j$, $\forall i, j$.*

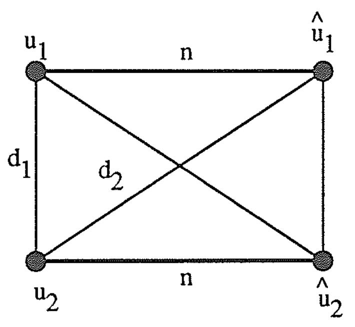


Figure 3.11: A butterfly of two *complementary* pairs of convolutional codewords.

Let $(u_1, \hat{u}_1)$, and $(u_2, \hat{u}_2)$ be complementary pairs of convolutional codewords of length n. Consider the **butterfly** pattern in Fig. 3.11. Let $d_1 = d(u_1, u_2)$ and $d_2 = d(u_2, \hat{u}_1)$. Then it is obvious that

$$d_1 + d_2 = d(u_1, \hat{u}_1) = n. \tag{3.6}$$

Because the above equality holds for any complementary pair $(u_1, \hat{u}_1)$ and any u_2, it follows by interchanging the roles of u_1 and u_2 that

$$d_1 + d(u_1, \hat{u}_2) = n. \tag{3.7}$$

From the equations (3.6) and (3.7), $d_2 = d(u_1, \hat{u}_2)$, and in general

$$\begin{aligned} d(u_1, u_2) &= d(\hat{u}_1, \hat{u}_2) \\ d(u_1, \hat{u}_2) &= d(\hat{u}_1, u_2) \end{aligned} \tag{3.8}$$

i.e., the weights of the side edges or the cross edges are identical. This property contributes to the cross-symmetry of the standard Hamming matrix D_u.

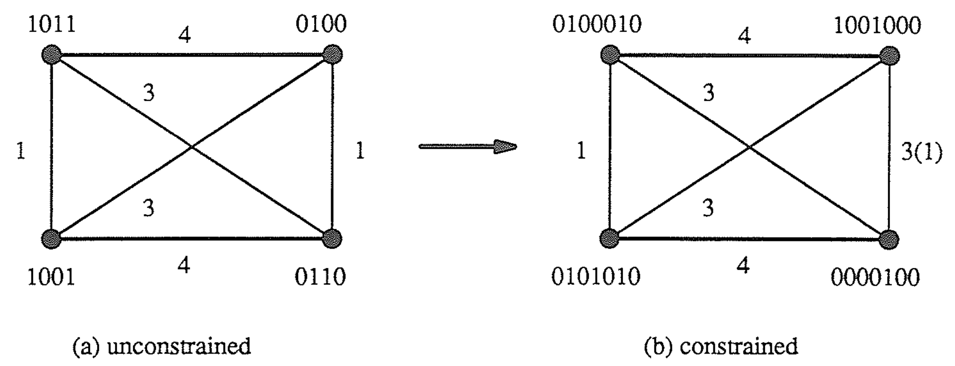


Figure 3.12: Butterfly patterns with $d = 1$, $k/n = 3/4$, and $k/l = 3/7$.

However, this symmetry need not hold for a constrained counterpart, as can be seen below. Consider Fig. 3.12(b) which shows two matched pairs of constrained codewords of 7 bits in length, $(v_1, \hat{v}_1) = (0100010, 1001000)$, and $(v_2, \hat{v}_2) = (0101010, 0000100)$. The distance pair in the given constrained butterfly is not symmetric—$d(v_1, v_2) = 1$ and $d(\hat{v}_1, \hat{v}_2) = 3$. However these pairs $(v_i, \hat{v}_i)$, $i = 1, 2$, can replace the convolutional codeword pairs of Fig. 3.12(a), since the corresponding Hamming distances are preserved. Note that the butterfly of Fig. 3.12(a) is the only kind which can be replaced by that of Fig. 3.12(b). Therefore the distance $d(\hat{v}_1, \hat{v}_2) = 3$ in (b) can be regarded to be equivalent to the corresponding distance $d(\hat{u}_1, \hat{u}_2) = 1$ in (a), *i.e.*, it can be reduced to one.

This reducing or balancing of edge weights has the important effect of making the constrained distance matrix cross-symmetric. In other words, for the purpose of Hamming maps, the butterfly weights can be reduced so that the distance matrix becomes cross-symmetric.

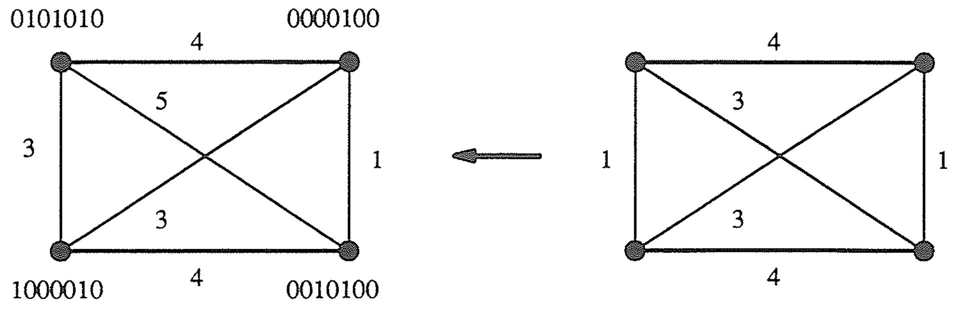


Figure 3.13: A strongly balanced butterfly: $d = 1$, $k/n = 3/4$, and $k/l = 3/7$.

Various forms of weight balancing are illustrated in Fig. 3.13 through Fig. 3.15. Consider a butterfly in Fig. 3.13 whose balanced weights on the cross edges satisfy the equality (3.6). When this is the case, the balanced butterfly is termed *strongly balanced.*

On the other hand, the butterfly in Fig. 3.14 cannot be balanced uniquely, as above. The unbalanced weights already satisfy the symmetry of Eq. 3.8. It is not certain which butterfly can be replaced by these pairs of matched codewords. This kind is termed *weakly balanced.* The butterfly of Fig. 3.15 has different weights on the matching edges. Balancing the different kinds of butterflies is based on the following rules.

R1 $d(v, \hat{v}) = n$.
R2 $d_1 = \min(d(v_1, v_2), d(\hat{v}_1, \hat{v}_2)) < n$
$d_2 = \min(d(v_1, \hat{v}_2), d(\hat{v}_1, v_2)) < n$
R3 $d_1 + d_2 = n + c$, for an even value c.

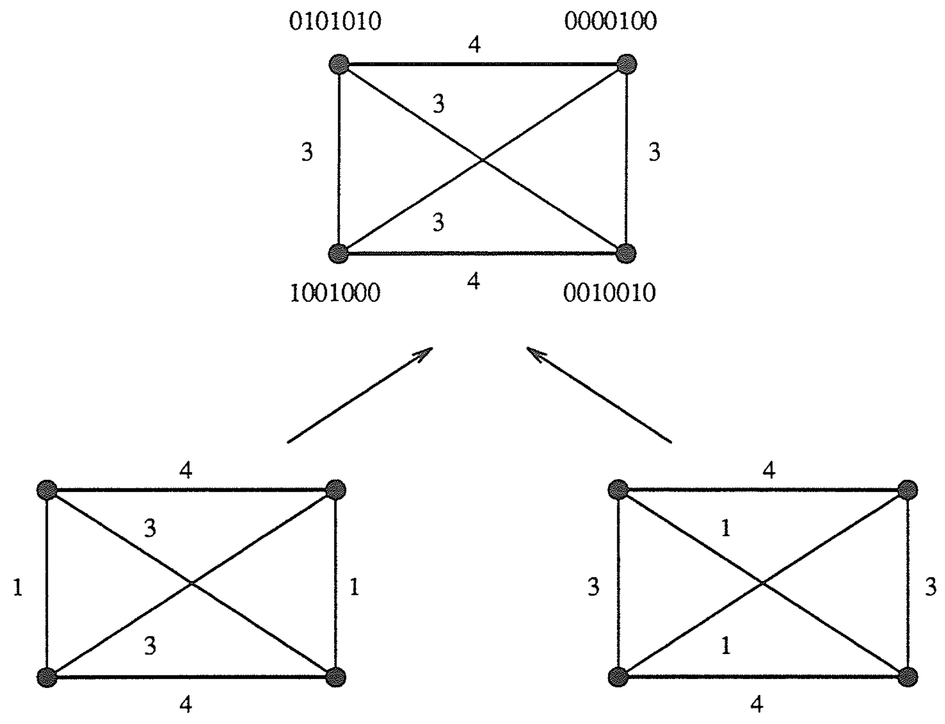


Figure 3.14: A weakly balanced butterfly: $d = 1$, $k/n = 3/4$, and $k/l = 3/7$.

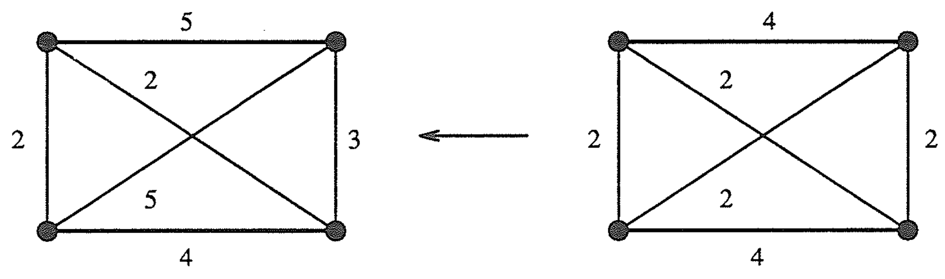


Figure 3.15: A strongly balanced butterfly (mixed matching edges): $d = 1$, $k/n = 3/4$, and $k/l = 3/7$.

3.3.2 Canonical Ordering

The *canonical ordering algorithm* is used to place a set of codeword pairs into a set of slot pairs subject to a restriction termed *adjacency*. Necessary terms and symbols are given first.

Suppose $M = \{e'_0, e'_1, \dots, e'_{m-1}\}$, $m \geq q/2$, is a maximum matching of G. Denote a subset of $q/2$ edges (q vertices) as H (V):

$$\begin{aligned} H &= \{e_0, e_1, \dots, e_{q/2-1}\}, \\ V &= \{(c_0, \hat{c}_0), (c_1, \hat{c}_1), \dots, (c_{q/2-1}, \hat{c}_{q/2-1})\}, \ e_i = (c_i, \hat{c}_i) \in M. \end{aligned}$$

Assume that the Hamming distance matrix D_v has been balanced according to rules **R1–R3**.

Definition 3.2 (Slot map) *Let I denote the set of slots, or the domain of a Hamming map.*

$$I = \{s_0, s_1, \dots, s_{q-1}\}, \ 0 \leq s_i \leq q-1.$$

Call the function $h : I \longrightarrow V$, whose value $h(s)$ denotes the codeword occupying a slot s, a slot map. Similarly, the inverse map $g : V \longrightarrow I$ is called a codeword map, whose value $g(v)$ represents the slot occupied by a codeword v:

The *adjacency* of codewords and slots plays a vital role in the *canonical ordering* algorithm.

Definition 3.3 (Adjacency) *A pair of codewords (v_1, v_2) are called d-*adjacent *to each other if $d(v_1, v_2) = d$. On the other hand, a pair of slots (s_1, s_2)*

Table 3.4: Adjacent slots $\Lambda(s)$ when $n = 2, 3$.

s	s_0	s_1
0	1	2
1	0	3
2	0	3
3	1	2

s	s_0	s_1	s_2
0	3	5	6
1	2	4	7
2	1	4	7
3	0	5	6
4	1	2	7
5	0	3	6
6	0	3	5
7	1	2	4

are called d-adjacent *to each other if* $d^u_{s_1,s_2} = d$ *in the unconstrained distance matrix* D_u.

It should be noted that adjacency on *slots* applies to the *unconstrained* case while adjacency of *codewords* applies to the *constrained* case.

Because the balanced matrix is both symmetric and cross-symmetric, the codeword pair (v_1, v_2) are d-adjacent if and only if $(\hat{v}_1, \hat{v}_2)$ are d-adjacent to each other. Let $\Lambda(s)$ be the set of $(n-1)$-adjacent slots to a slot $s \in I$ and $\Gamma(v)$ the set of $(n-1)$-adjacent codewords to a codeword $v \in V$.

$$\begin{aligned} \Lambda(s) &= \{s_0, s_1, \ldots, s_{n-1}\} \\ \Gamma(v) &= \{v_0, v_1, \ldots, v_{t-1}\} \end{aligned}$$

Note that the cardinality of set $\Lambda(s)$ equals $\begin{pmatrix} n \\ n-1 \end{pmatrix}$, the $(n-1)$-th degree δ^{n-1}_u for an unconstrained codeword $u \in U$. The necessary condition NC3 requires that the cardinality t of $\Gamma(v, n-1)$ be n or greater for v to be a member of a Hamming map. Table 3.4 and 3.5 shows $(n-1)$-adjacent slots for $n = 2, 3$, and 4.

The canonical ordering is greatly facilitated by the following theorem.

Table 3.5: Adjacent slots $\Lambda(s)$ when $n = 4$.

s	s_0	s_1	s_2	s_3	s	s_0	s_1	s_2	s_3
0	7	11	13	14	8	3	5	6	15
1	6	10	12	15	9	2	4	7	14
2	5	9	12	15	10	1	4	7	13
3	4	8	13	14	11	0	5	6	12
4	3	9	10	15	12	1	2	7	11
5	2	8	11	14	13	0	3	6	10
6	1	8	11	13	14	0	3	5	9
7	0	9	10	12	15	1	2	4	8

Theorem 3.1 (Canonical matrix: unconstrained) *Suppose that each matched codeword pair $(v, \hat{v})$ are placed in a matched slot (row) pair such that all entries of D_u whose value is $(n-1)$ fall in the $(n-1)$-adjacent slots. Then the resulting matrix is a* canonical *matrix, which corresponds to the set of unconstrained codewords arranged in natural order.*

Proof The proof uses a recursive statement based on the size q of matrix D_q. Suppose that all $(n-1)$-adjacent codewords are in $(n-1)$-adjacent slots. It follows from the triangle equality of (3.6) and the symmetry (3.8) that all 1-adjacent codewords are in their 1-adjacent slots.

Write the matrix D_q as a 2×2 block matrix:

$$D_q = \begin{bmatrix} D_{q/2} & \overbrace{D_{q/2}} \\ \overbrace{D'_{q/2}} & D'_{q/2} \end{bmatrix}$$

The main cross-diagonal entries of $D_{q/2}$ and $D'_{q/2}$ are $n-1$ since their corresponding slots are $(n-1)$-adjacent in D_q. As a consequence, in each submatrix $D_{q/2}$ and $D'_{q/2}$, all $(n-2)$'s are in the $(n-2)$-adjacent slots, and thus all 2's are in the 2-adjacent slots of D_q again by 3.6 and 3.8. That is, a correct placement of the $(n-1)$'s of D_q forces the correct placement of the $(n-2)$'s of $D_{q/2}$ and $D'_{q/2}$. The same argument applys to the submatrices $D_{q/2}$, $D_{q/4}$, etc., until the submatrix becomes D_2, which is the identity matrix.

To demonstrate the usefulness of the theorem, the following procedure unscrambles a set of mixed codewords into a Hamming map.

Example: Unscrambling codewords

Consider a set of 8 convolutional codewords (4 matched pairs) which are scrambled in an arbitrary way:

$$\{(c_0, \hat{c}_0), (c_1, \hat{c}_1), (c_2, \hat{c}_2), (c_3, \hat{c}_3)\} \tag{3.9}$$

where $\hat{c}_i = c_{\hat{i}}$, $\hat{i} = q-1-i = 7-i$, $0 \leq i \leq q-1 = 7$. The distance matrix D_v for these codewords are given in Table 3.6. Note that each codeword is a 3-sequence, *i.e.*, it passes the necessary test NC3. Note also that the main cross entries $(d_{i\hat{i}}) = d(c_i, \hat{c}_i) = n = 3$. The matrix, however, is not in canonical form yet.

The *task* is to determine whether a Hamming map exists and, if the answer is yes, to find a proper arrangement of these 8 codewords so that their matrix is canonical.

Procedure: All the 8 slots $\{(s_0, \hat{s}_0), (s_1, \hat{s}_1), (s_2, \hat{s}_2), (s_3, \hat{s}_3)\}$ are empty initially. Choose a pair of matched codewords from Table 3.6, say, $(c_5, \hat{c}_5) =$

Table 3.6: An 8×8 distance matrix D_v.

V	$\hat{V}$		c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7
c_0	$\hat{c}_7$	010	0	2	1	1	2	2	1	3
c_1	$\hat{c}_6$	111	2	0	1	1	2	2	3	1
c_2	$\hat{c}_5$	011	1	1	0	2	1	3	2	2
c_3	$\hat{c}_4$	110	1	1	2	0	3	1	2	2
$\hat{c}_3$	c_4	001	2	2	1	3	0	2	1	1
$\hat{c}_2$	c_5	100	2	2	3	1	2	0	1	1
$\hat{c}_1$	c_6	000	1	3	2	2	1	1	0	2
$\hat{c}_0$	c_7	101	3	1	2	2	1	1	2	0

(c_5, c_2). Place this pair of codewords into a matched slot pair, say, $(s_0, \hat{s}_0)$, *i.e.*,

$$h(s_0) = c_5,\ h(\hat{s}_0) = h(s_7) = c_2.$$

This placement leaves 6 slots unoccupied.

s	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
$h(s_0)$	c_5							c_2

The 2-adjacent slots of s_0 and 2-adjacent codewords of c_5 are determined.

$$\Lambda(s_0) = \{s_3, s_5, s_6\} \quad \text{[Table 3.4]}$$

$$\Gamma(c_5) = \{c_0, c_1, c_4\} \quad \text{[Table 3.6]}$$

Placement of the codewords of $\{\Gamma(c_5), \Gamma(\hat{c}_5)\}$ into the slots of $\{\Lambda(s_0), \Lambda(\hat{s}_0)\}$, **in any order**, accomplishes the desired arrangement. One such placement is shown below:

s	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
$h(s_0)$	c_5							c_2
$h(\Lambda(s_0))$				c_0		c_1	c_4	
$h(\Lambda(\hat{s}_0))$		c_3	c_6		c_7			

The resultant matrix, as shown in Table 3.7 is indeed a canonical matrix and a 1:1 map $f : U \rightarrow V$, where

$$\begin{aligned} U &= (000, 001, 010, 011, 100, 101, 110, 111) \\ V &= (c_5, c_3, c_6, c_0, c_7, c_1, c_4, c_2), \end{aligned}$$

is a Hamming map.

The number of steps required for unscrambling 8 codewords was 1, in sharp contrast to $8! = 40320$, which an exhaustive trial may require. In general, $2^n/8$ steps are required for unscrambling 2^n convolutional codewords into a canonical form, since each step will effectively assign 4 pairs of codewords at a time.

To extend this technique to the constrained case, further terms need to be defined. Let p be a slot called a *pivot* and $v = h(p)$. Suppose

$$\begin{aligned} \Lambda(p) &= \{s_0, s_1, \ldots, s_{n-1}\}, \text{ and} \\ \Gamma(v) &= \{v_0, v_1, \ldots, v_{t-1}\}, \ t \geq n. \end{aligned}$$

Definition 3.4 (Assignment graph) *Let $G = (X, Y)$ be a bipartite graph termed an* assignment graph. *The vertex set $V(X, Y)$ is*

$$X = \{X_0, X_1, \ldots, X_{n'-1}\}, \ n' \leq n, \ \textit{and}$$

Table 3.7: An 8 × 8 canonical distance matrix D_v.

U	V		c_5	c_3	c_6	c_0	c_7	c_1	c_4	c_2
u_0	c_5	100	0	1	1	2	1	2	2	3
u_1	c_3	110	1	0	2	1	2	1	3	2
u_2	c_6	000	1	2	0	1	2	3	1	2
u_3	c_0	010	2	1	1	0	3	2	2	1
u_4	c_7	101	1	2	2	3	0	1	1	2
u_5	c_1	111	2	1	3	2	1	0	2	1
u_6	c_4	001	2	3	1	2	1	2	0	1
u_7	c_2	011	3	2	2	1	2	1	1	0

$$Y \ = \ \{Y_0, Y_1, \ldots, Y_{t'-1}\}, \ t' \leq t,$$

where X_i is the set of assigned codewords in filled slots that are $(n-1)$-adjacent to the i-th empty slot s_i of $\Lambda(p)$, and Y_i is the set of assigned codewords that are $(n-1)$-adjacent to the i-th non-assigned codeword v_i of $\Gamma(v)$. The vertices X_i and Y_j are joined if and only if X_i is contained in Y_j, i.e., *$e = (X_i, Y_j)$ is an edge of $E(X, Y)$ if and only if the codeword v_j is qualified to take the slot s_i.*

A maximum matching $A = \{(X_{i'}, Y_{i'})\}$ of the bipartite graph $G = (X, Y)$ defines a possible assignment of codeword pairs $(v_{i'}, \hat{v}_{i'})$ of $\Gamma(v)$ into slot pairs $(s_{i'}, \hat{s}_{i'})$ of $\Lambda(p)$. This assignment is based on the reasoning that for a codeword v to be placed into a slot s, the codeword v should be at distance $(n-1)$ from other codeword w, if w has already taken a slot of $\Lambda(s)$.

In contrast to the standard matrix D_u of convolutional codewords, the *balanced* matrix D_v of constrained codewords has both *strongly* and *weakly* balanced butterflies. This fact prevents the canonical ordering algorithm from finding a Hamming map in $2^n/8$ steps, the number of steps required for unscrambling convolutional codewords into a standard form. Thus for the constrained case, each maximum matching (assignment) from the assignment graph has to be examined in general. The following algorithm reflects this.

Algorithm Canonical Ordering

```
Procedure CanOrder(p: SlotType): Boolean;
  {
    var
      G : Bipartite graph;
      A : Assignment;
      v : CodewordType;
      p : SlotType;
    if all slots have been filled return completed;
    $v := h(p)$;
    build the assignment graph $G := (X, Y)$ for $(\Gamma(v), \Lambda(p))$ by Def. 3.3;
    find an assignment A of G; {by Hungarian maximum matching}
    if $|A| < |X|$ return not completed;
    for each assignment A of G {
      do assignment A;
      choose new pivot p; {say, every 4-th slot from $s_0$}
      if (CanOrder(p)) = completed return completed;
      else cancel assignment A;
    } {end for}
    return not completed;
  } {end CanOrder}

Procedure Main;
  {
    var
      $c, \hat{c}$ : CodewordType;
      $p, \hat{p}$ : SlotType;
    pick $(c, \hat{c})$;
    initialize pivot; {$p := s_0$ and $\hat{p} := s_{q-1}$}
    assign $(c, \hat{c})$ into $(p, \hat{p})$;
    if ( CanOrder(p) = completed ) save Hamming map;
    else signal "no Hamming map exists";
  } {end Main}
```

This algorithm can be improved by trying *strongly* adjacent codewords first for assignment and then *weakly* adjacent codewords if necessary.

Definition 3.5 (Strength) *Assume that the distance matrix has been balanced. Divide the codewords of* $\Gamma(v)$ *into two sets of* $(n-1)$*-adjacent codewords,* $\Gamma(v) = S(v) \cup W(v)$*, where*

$$\begin{aligned} S(v) &= \{w \in \Gamma(v) \mid d(v,w) + d(\hat{v},w) = n\}, \\ W(v) &= \{w \in \Gamma(v) \mid d(v,w) + d(\hat{v},w) > n\}. \end{aligned}$$

Note that pairs $(v,\hat{v})$ *and* $(w,\hat{w})$ *form a strongly balanced butterfly if* w *is in* $S(v)$*, and* w *is termed* strongly $(n-1)$*-adjacent to* v*. Similarly,* $w \in W$ *is termed* weakly $(n-1)$*-adjacent to* v*.*

Note that each strongly adjacent codeword should be assigned from the assignment graph. Otherwise weakly adjacent codewords will occupy the slots reserved for strongly adjacent codewords. This in turn will result in a violation of NC3 due to extra 1 entries from one side of the *missed* strongly balanced butterflies.

The example in the following section, which is rather long and complicated, has been designed to trace the details of the assignment procedure. An excessive number of tables and figures are presented to indicate the labour involved in finding a Hamming map.

3.3.3 An example: canonical arrangement

<u>1. Parameters</u>:

$d = 1, r_u = k/n = 3/4, r_c = k/l = 3/7, N_d(l) = 21, p = 20, m = 9$. The symbol p denotes the number of codewords which have passed test NC3 and m the number of edges in a maximum matching M of constrained graph G_v.

2. A subset of an m-matching M:

Consider the set of 8 pairs of codewords as shown in Table 3.8. Each pair $(c_i, c_{\hat{i}})$, $\hat{i} = 15 - i$, having weight $n = 4$, represents a matching edge in M. The *Edmonds* maximum matching algorithm has been used to find an m-matching with $m = 9$, out of which 8 edges have been chosen randomly. The Table 3.8 lists $(n-1)$-adjacent codewords as a union of strong and weak sets from each codeword c. Note that the constrained matrix has been balanced before making the decision on adjacency.

3. Canonical Ordering:

STEP 0:

It shall be convenient to use $h(s, \hat{s}) = (v, \hat{v})$ to mean the assignment of a pair of codewords into a pair of slots at the same time, *i.e.*, $h(s) = v$, $h(\hat{s}) = \hat{v}$. Choose a pair of codewords, say, (c_3, c_{12}). Because no slot has been oc-

Table 3.8: $(n-1)$-adjacent codewords $\Gamma(c), n = 4$.

c	S				W				hex	binary
0	7	11	13	14					0A	0001010
1	10	12	15		6	9			2A	0101010
2	5	9	15		3	12			08	0001000
3	8	14			2	4	11	13	22	0100010
4	9	10	15		3	12			4A	1001010
5	2	8	11	14					42	1000010
6	11	13			1	7	8	14	48	1001000
7	0	10	12		6	9			20	0100000
8	3	5	15		6	9			54	1010100
9	2	4			1	7	8	14	12	0010010
10	1	4	7	13					14	0010100
11	0	5	6		3	12			10	0010000
12	1	7			2	4	11	13	44	1000100
13	0	6	10		3	12			52	1010010
14	0	3	5		6	9			04	0000100
15	1	2	4	8					24	0100100

cupied yet, the codeword pair can be freely assigned into any slot pair, say, $(s_0, \hat{s}_0)$, with adjacency considerations ignored.

$$h(s_0, \hat{s}_0) = h(s_0, s_{15}) = (c_3, c_{12})$$

Call the slot pair $p = s_0$ and $\hat{p} = \hat{s}_0$ a *pivot* slot pair. Observe that all slots but the *pivot slot* pair $(p, \hat{p})$ are left unoccupied.

STEP 1:

Consider the $(n-1)$-adjacent slots $\Lambda(p)$ of the pivot slot p. These slots together with their matched slots $\Lambda(\hat{p})$ are to be occupied by the $(n-1)$-adjacent codewords $\Gamma(v)$, and $\Gamma(\hat{v})$, respectively. From Table 3.5 and Table 3.8,

$$\begin{aligned} \Lambda(p) = \Lambda(s_0) &= \{s_7, s_{11}, s_{13}, s_{14}\} \qquad (3.10) \\ \Gamma(v) = \Gamma(c_3) &= \{w_0, w_1, \ldots\} = \{c_8, c_{14}\} + \{c_2, c_4, c_{11}, c_{13}\} = S \cup W. \end{aligned}$$

These adjacency relationships are depicted in Fig. 3.16. All the slots $\Lambda(p)$ are empty, and therefore the *assignment graph* $G = (X, Y)$ for **step 1** becomes a complete bipartite graph K_{nt} which has $n \times t$ edges, where $t = \mid \Gamma(v) \mid$. This yields too many assignments for examination.

For this **initial step**, however, it should be noted that the strongly adjacent codewords in S can take any empty slots. To exploit this fact, the codewords in S are assigned arbitrarily to obtain an assignment. The graph based on weak set W, (very) much smaller than K_{nt}, may then be considered if necessary.

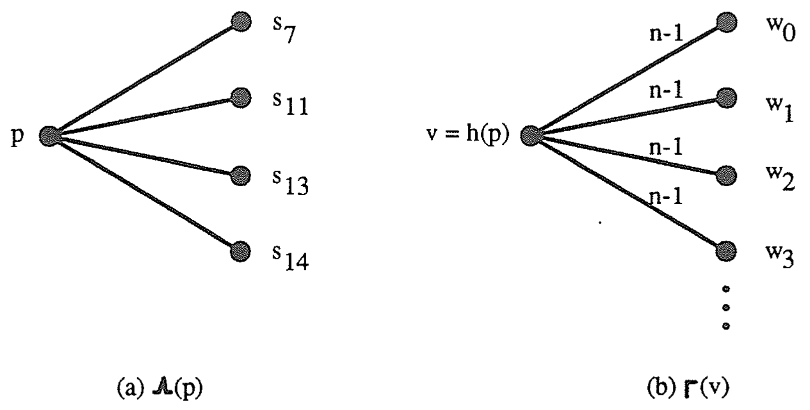


Figure 3.16: Adjacency for pivot $p = s_0$.

Since the assignment of strongly adjacent codewords is arbitrary, the graph shown in Fig. 3.17(a) need not be constructed at all. As the assignment is not complete, each maximum matching (assignment) from graph $G(W)$ of Fig. 3.17(b) has to be examined for a possible codeword assignment to empty slots s_7 and s_{11}. Note that $G(W)$ has 8 edges, and is much smaller than the 24 edges of the whole graph G. In terms of number of maximum matching, the comparison becomes significant, 12 versus 360.

An assignment $A(s_0) = \{(s_7, c_2), (s_{11}, c_{11})\}$ corresponding to a maximum matching of $G(W)$ in Fig. 3.17(b), is shown in Table 3.9, which also compiles all the assignments made to this point—10 out of 16 slots in a single step !!!. The symbols, spade-heart (♠,♡), mark the adjacent slots $(s, \hat{s})$ for $s \in \Lambda(p)$.

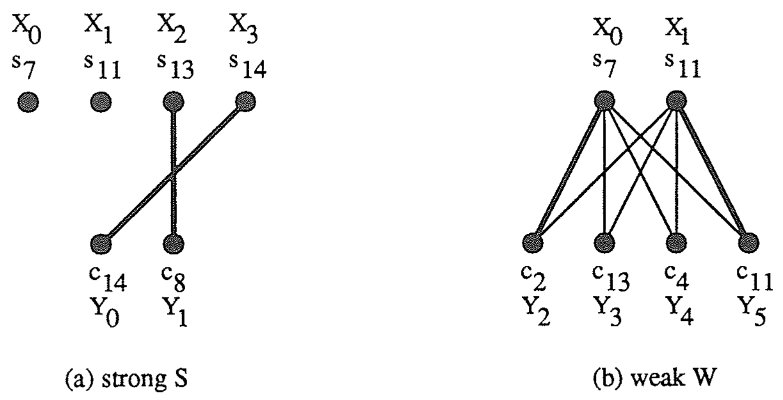


Figure 3.17: An assignment $A(s_0)$ of graphs $G(S)$ and $G(W)$ for pivot $p = s_0$.

Table 3.9: An assignment $A(s_0)$ from pivot $p = s_0$.

s	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
$\Lambda(s_0)$	c_3	♡	♡		♡			♠	♡			♠		♠	♠	c_{12}
$A(s_0)$		c_1	c_7		c_4			c_2	c_{13}			c_{11}		c_8	c_{14}	

STEP 2:

Note that each of the unoccupied slot pairs is $(n-1)$-adjacent to the slot pair $(s_4, \hat{s}_4)$. With this in mind, choose a new pivot $p = s_4$ and set $v = h(p)$. A similar process to that in **step 1** is performed with the new pivot.

In contrast to the initial step, some slots are already filled and some codewords have been used (assigned). In this step the construction of the assignment graph is not a simple task. See the schematic assignment graph in Fig.3.18.

The task is to put available codewords $\{c_9, c_{10}, c_{15}\}$ into slots $\{s_3, s_9, s_{10}\}$ such that the resulting arrangement conforms to the adjacency consideration. First the assignment graph is constructed as described in Definition 3.3:

$$\Lambda(p) = \Lambda(s_4) = \{s_3, s_9, s_{10}, \overline{s_{15}}\} \tag{3.11}$$

$$\Gamma(v) = \Gamma(c_4) = \{v_0, v_1, \ldots\} = \{c_9, c_{10}, c_{15}\} + \{\overline{c_3}, \overline{c_{12}}\} = S \cup W,$$

where the slot s_{15} has been filled, and the codewords $\{c_3, c_{12}\}$ have been used (assigned) by the assignment $A(s_0)$ for pivot s_0. These adjacency relationships are depicted in Fig. 3.19.

With the aid of Table 3.5 and Table 3.8, the bipartition (X, Y) can be determined to be:

$$\begin{aligned}
X &= \{X_0, X_1, X_2\} \\
X_0 &= \text{codewords of } \Lambda(s_3) = \{h(s_4), h(s_8), h(s_{13}), h(s_{14})\} = \{c_4, c_{13}, c_8, c_{14}\} \\
X_1 &= \text{codewords of } \Lambda(s_9) = \{h(s_2), h(s_4), h(s_7), h(s_{14})\} = \{c_7, c_4, c_2, c_{14}\}
\end{aligned}$$

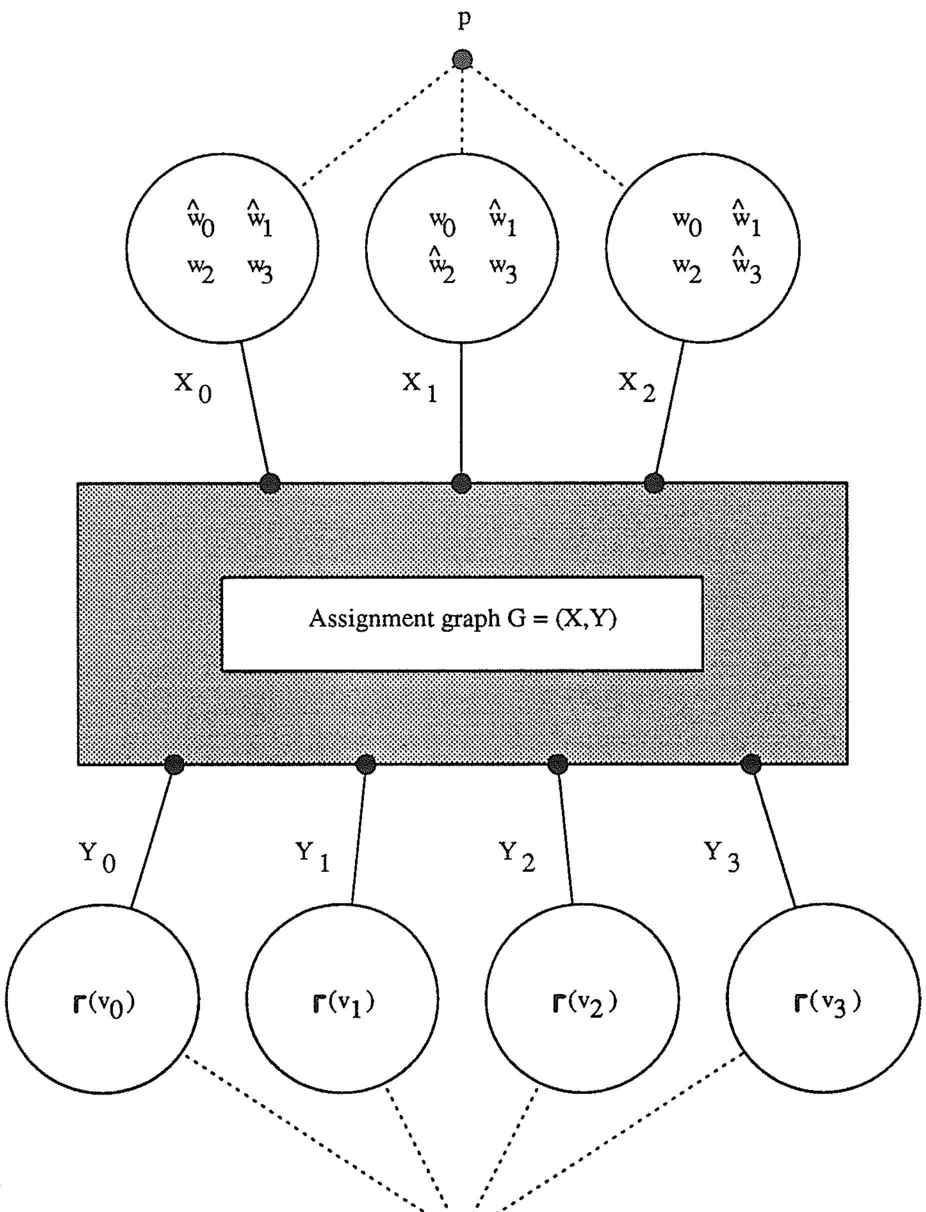


Figure 3.18: Assignment graph $G = (X, Y)$ for pivot $p = s_4$.

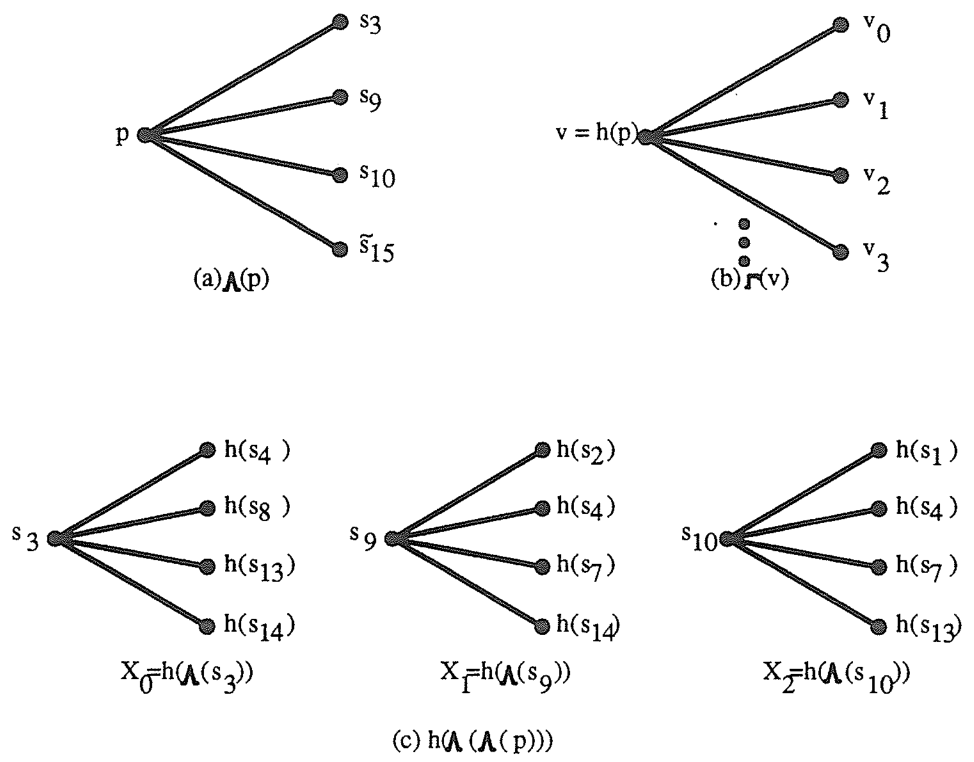


Figure 3.19: Adjacency for pivot $p = s_4$.

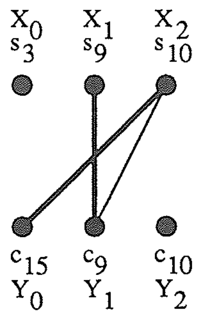


Figure 3.20: An incomplete assignment $A(s_4)$ for pivot $p = s_4$, leaving the slot pair $(s_3, \hat{s}_3)$ unoccupied.

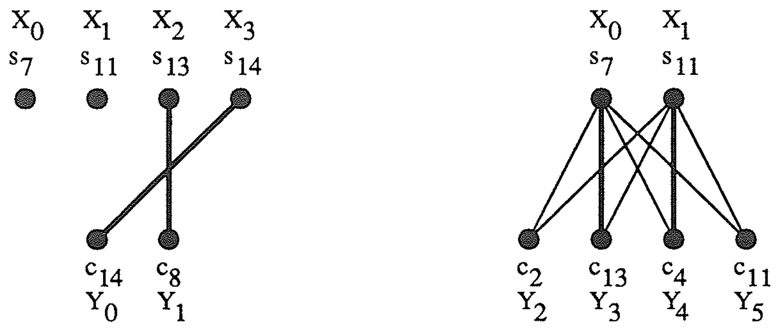
$$\begin{aligned} X_2 &= \text{codewords of } \Lambda(s_{10}) = \{h(s_1), h(s_4), h(s_7), h(s_{13})\} = \{c_1, c_4, c_2, c_8\} \\ Y &= \{Y_0, Y_1, Y_2\} \\ Y_0 &= \text{assigned } \Gamma(c_{15}) = \{c_1, c_2, c_4, c_8\}. \\ Y_1 &= \text{assigned } \Gamma(c_9) = \{c_1, c_2, c_4, c_7, c_8, c_{14}\} \\ Y_2 &= \text{assigned } \Gamma(c_{10}) = \{c_1, c_4, c_7, c_{13}\} \end{aligned}$$

The set relations are: $X_1 \subset Y_1$, $X_2 \subset Y_0$, $X_2 \subset Y_1$. As an example codewords (c_9, c_{15}) can take slots (s_9, s_{10}). The assignment graph for this adjacency relation is shown in Fig. 3.20. Note that the assignment graph is no longer a complete bipartite graph. This results in a reduced number of possible assignments, a *highly desirable feature.* It can be seen immediately that slots $(s_3, \hat{s}_3)$ cannot be assigned at this stage. Table 3.10 shows an incomplete assignment $A(s_4)$.

Two possibilities can be considered: a Hamming map does not exist, or the assignment $A(s_0)$ in the previous step is incorrect. To see if $A(s_0)$ is

Table 3.10: An incomplete assignment $A(s_4)$ from pivot s_4 resulting from $A(s_0)$: the slot pair (s_3, s_{12}), marked as □, are not filled.

s	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
$\Lambda(s_0)$	c_3	♡	♡		♡			♠	♡			♠		♠	♠	c_{12}
$A(s_0)$		c_1	c_7		c_4			c_2	c_{13}			c_{11}		c_8	c_{14}	
$\Lambda(s_4)$				♠	c_4	♡	♡			♠	♠	c_{11}	♡			
$A(s_4)$				□		c_0	c_6			c_9	c_{15}		□			



(a) strong S (b) weak W

Figure 3.21: An alternate assignment $A'(s_0)$ of $G(W)$ for pivot $p = s_0$.

incorrect, algorithm retreats to **step 1** and examines alternate assignments.

STEP 1′ (retreat from STEP 2):

This time, as depicted in Fig. 3.21 and Table 3.11, a new assignment $A'(s_0)$ is employed: $A'(s_0) = \{(s_7, c_{13}), (s_{11}, c_4)\}$

STEP 2′ (back to pivot s_4):

Table 3.11: An alternate assignment $A'(s_0)$ from pivot $p = s_0$.

s	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
$\Lambda(s_0)$	c_3	♡	♡		♡			♠	♡			♠		♠	♠	c_{12}
$A'(s_0)$		c_1	c_7		c_{11}			c_{13}	c_2			c_4		c_8	c_{14}	

Now $v = h(p) = c_{11}$ and the assignment graph G' is based on the alternate assignment $A'(s_0)$.

$$\Lambda(p) = \Lambda(s_4) = \{s_3, s_9, s_{10}, \overline{s_{15}}\} \tag{3.12}$$

$$\Gamma(v) = \Gamma(c_{11}) = \{v_0, v_1, \ldots\} = \{c_0, c_6, c_5\} + \{\overline{c_3}, \overline{c_{12}}\} = S \cup W.$$

$$\begin{array}{rcl}
X & = & \{X_0, X_1, X_2\} \\
X_0 & = & \text{codewords of } \Lambda(s_3) = \{h(s_4), h(s_8), h(s_{13}), h(s_{14})\} = \{c_{11}, c_2, c_8, c_{14}\} \\
X_1 & = & \text{codewords of } \Lambda(s_9) = \{h(s_2), h(s_4), h(s_7), h(s_{14})\} = \{c_7, c_{11}, c_{13}, c_{14}\} \\
X_2 & = & \text{codewords of } \Lambda(s_{10}) = \{h(s_1), h(s_4), h(s_7), h(s_{13})\} = \{c_1, c_{11}, c_{13}, c_8\} \\
Y & = & \{Y_0, Y_1, Y_2\} \\
Y_0 & = & \text{assigned } \Gamma(c_0) = \{c_7, c_{11}, c_{13}, c_{14}\} \\
Y_1 & = & \text{assigned } \Gamma(c_6) = \{c_1, c_7, c_8, c_{11}, c_{13}, c_{14}\} \\
Y_2 & = & \text{assigned } \Gamma(c_5) = \{c_2, c_8, c_{11}, c_{14}\}
\end{array}$$

The set relations are:

$$X_0 \subset Y_2,\ X_1 \subset Y_0,\ X_1 \subset Y_1,\ X_2 \subset Y_1.$$

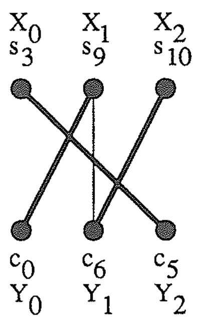


Figure 3.22: A complete assignment $A'(s_4)$ for pivot $p = s_4$.

The assignment graph G' for this adjacency relation is shown in Fig. 3.22. Now there is an assignment $A'(s_4)$ which assigns the slots completely:

$$A'(s_4) = \{(s_3, c_5), (s_9, c_0), (s_{10}, c_6)\}$$

The overall assignment $A(s)$ accomplished through **step 1′** and **step 2′** is shown in Table 3.12.

$$A(s) = A'(s_0) + A'(s_4)$$

The **canonical ordering** algorithm thus took 2 steps to find a Hamming map.

To check whether it is indeed a Hamming map, the corresponding distance matrix is shown below.

Table 3.12: A complete assignment $A(s)$, consisting of $A'(s_0)$ and $A'(s_4)$.

s	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
$\Lambda(s_0)$	c_3	♡	♡		♡			♠	♡			♠		♠	♠	c_{12}
$A'(s_0)$		c_1	c_7		c_{11}			c_{13}	c_2			c_4		c_8	c_{14}	
$\Lambda(s_4)$				♠	c_{11}	♡	♡			♠	♠	c_4	♡			
$A'(s_4)$				c_5		c_9	c_{15}			c_0	c_6		c_{10}			
$A(s)$	c_3	c_1	c_7	c_5	c_{11}	c_9	c_{15}	c_{13}	c_2	c_0	c_6	c_4	c_{10}	c_8	c_{14}	c_{12}

Constrained canonical distance matrix D_v

```
h[ 0]: 3 [22]  0 1 1 2  3 2 2 3  3 2 4 3  4 5 3 4
h[ 1]: 1 [2A]  1 0 2 3  4 3 3 4  2 1 3 2  5 6 4 5
h[ 2]: 7 [20]  1 2 0 3  2 3 1 4  2 3 3 4  3 4 2 3
h[ 3]: 5 [42]  2 3 3 0  3 2 4 1  3 2 2 1  4 3 3 2

h[ 4]:11 [10]  3 4 2 3  0 1 3 2  2 3 3 4  1 2 2 3
h[ 5]: 9 [12]  2 3 3 2  1 0 4 1  3 2 4 3  2 3 3 4
h[ 6]:15 [24]  2 3 1 4  3 4 0 5  3 4 4 5  2 3 1 2
h[ 7]:13 [52]  3 4 4 1  2 1 5 0  4 3 3 2  3 2 4 3

h[ 8]: 2 [08]  3 2 2 3  2 3 3 4  0 1 1 2  3 4 2 3
h[ 9]: 0 [0A]  2 1 3 2  3 2 4 3  1 0 2 1  4 5 3 4
h[10]: 6 [48]  4 3 3 2  3 4 4 3  1 2 0 1  4 3 3 2
h[11]: 4 [4A]  3 2 4 1  4 3 5 2  2 1 1 0  5 4 4 3

h[12]:10 [14]  4 5 3 4  1 2 2 3  3 4 4 5  0 1 1 2
h[13]: 8 [54]  5 6 4 3  2 3 3 2  4 5 3 4  1 0 2 1
h[14]:14 [04]  3 4 2 3  2 3 1 4  2 3 3 4  1 2 0 1
h[15]:12 [44]  4 5 3 2  3 4 2 3  3 4 2 3  2 1 1 0
```

This example leads to several **important observations.**

1. If any element in a strong set can not be used, it means that a Hamming map does not exist.

2. When separate graphs $G(S)$ and $G(W)$ are used in turn, a great deal of computation is saved, especially in the initial step 1 from pivot s_0.

3. From **step 2**, assignment graphs generally have a very small number of edges, reflecting adjacency consideration from slots which have been assigned in earlier steps.

4. If every possible graph assignment from pivot s_0 has been tried without resulting in a complete assignment in the subsequent steps, it means that a Hamming map does not exist.

5. This process is repeated until all slots are completely assigned.

The improved algorithm based on these observations is outlined below. The procedure *CanOrder* is divided into two procedures: *Begin_CanOrder* for the initial pivot $p = s_0$ and *CanOrder* for pivots other than s_0.

In the procedure *Begin_CanOrder*, the strongly adjacent codewords of $S(v)$ are assigned arbitrarily (*no assignment graph built*), and the slots left unoccupied are tried by the weakly adjacent codewords of $W(v)$ with the aid of the assignment graph $G(W)$.

On the other hand, in the procedure *CanOrder*, an assignment graph $G(S)$ with the codewords of $S(v)$ is *constructed* and a maximum assignment $A(S)$ is sought first. When $A(S)$ fails to fill all the adjacent slots, another assignment graph $G(W)$ with the weakly adjacent codewords of $W(v)$ is *added* to the strong assignment graph $G(S)$, from which each assignment is examined.

```
Procedure *Main*;
 {
   var
     $c, \hat{c}$ : *CodewordType*;
     $p, \hat{p}$ : *SlotType*;
   pick $(c, \hat{c})$;
   initialize pivot; $\{p := s_0$ and $\hat{p} := s_{q-1}\}$
   assign $(c, \hat{c})$ into $(p, \hat{p})$;
   if $(Begin\_CanOrder(p))$ = completed save Hamming map;
   else signal “no Hamming map exists”;
 } {end *Main*}
```

Algorithm Initiate Canonical Ordering

```
Procedure $Begin\_CanOrder(s_0$: $SlotType)$: Boolean;
  {
    var
      $G(W)$ : $Bipartite\ graph$;
      $A(W)$ : $Assignment$;
      $v$ : $CodewordType$;
      $p$ : $SlotType$;
    $p := s_0, v := h(p)$;
    assign $S(v)$ into $\Lambda(p)$ arbitrarily;
    if $|S(v)| = |\Lambda(p)|$ {
      choose new pivot $p := s_4$;
      if ($CanOrder(p)$)=completed return completed;
      else signal "no Hamming map exists";
    } {end if}
    else { {assignment not complete}
      build assignment graph $G(W) = (X(W), Y(W))$;
      find an assignment $A(W)$ of $G(W)$;
      if $|A(W)| < |X(W)|$ return not completed;
      for each assignment $A(W)$ of $G(W)$ {
        do assignment $A(W)$;
        choose new pivot $p = s_4$;
        if ($CanOrder(p)$) = completed return completed;
        else cancel assignment $A(W)$;
      } {end for}
      return not completed;
    } {end else}
  } {end $Begin\_CanOrder$}
```

Algorithm Canonical Ordering

```
Procedure CanOrder(p: SlotType): Boolean;
  {
    var
      $G, G(S)$ : Bipartite graph;
      $A, A(S)$ : Assignment;
      $v$ : CodewordType;
      $p$ : SlotType;
    $p := s_0, v := h(p)$;
    build strong assignment graph $G(S) = (X(S), Y(S))$;
    find an assignment $A(S)$ of $G(S)$;
    if $|A(S)| = |X(S)|$ {
      for each assignment $A(S)$ of $G(S)$ {
        do assignment $A(S)$;
        choose new pivot $p$;{say, every 4-th slot from $s_0$}
        if (CanOrder(p)) = completed return completed;
        else cancel assignment $A$;
      } {end for}
      return not completed;
    } {end if}
    else {
      add $G(W)$ to $G(W)$ to form assignment graph $G := (X, Y)$;
      find an assignment $A$ of $G$;
      if $|A| < |X|$ return not completed;
      for each assignment $A$ of $G$ {
        do assignment $A$;
        choose new pivot $p$;
        if (CanOrder(p)) = completed return completed;
        else cancel assignment $A$;
      } {end for}
      return not completed;
    } {end else}
  } {end CanOrder}
```

3.3.4 Conjugate Hamming maps

Closely associated with one Hamming map is a class of maps termed *conjugate* Hamming maps. Once a Hamming map is found from the canonical ordering, $2^n \times n!$ conjugate Hamming maps can be obtained from it. Each of the conjugate maps use the same set of matching edges but arranged in different order and orientation. Of course, these could be found from the *canonical ordering* algorithm if one considers every possible assignment from strong and weak graphs and chooses different pivot slot pairs, or codeword pairs, etc. The following theorem shows that given a Hamming map, conjugate Hamming maps may be found analytically.

Theorem 3.2 (Conjugate maps) *Suppose that the mapping $f : U \to V$ defined by*

$$f : (u_0, u_1, \ldots, u_{q-1}) \to (v_0, v_1, \ldots, v_{q-1}), \;\; q = 2^n$$

is a Hamming map. Then a different Hamming map, termed a **conjugate map of** *f, can be obtained by a permutation operation P on the index set I:*

$$P : I = (0, 1, \ldots, q-1) \to (i_0, i_1, \ldots, i_{q-1}), \;\; i_j \in I.$$

The new Hamming map $f^* : U \to V \;\Leftrightarrow\; f : P(U) \to V$ is achieved by:

$$f : (u_{i_0}, u_{i_1}, \ldots, u_{i_{q-1}}) \to (v_0, v_1, \ldots, v_{q-1}).$$

The number C_n of such permutations P can be computed by the recursive equation:

$$C_n = 2n \cdot C_{n-1}, \;\; \text{with } C_1 = 2.$$

Proof: The proof is completed by showing that the associated distance matrices D_n and $\tilde{D}_n = P(D_n)$ with two q-tuples U and $P(U)$ are identical, *i.e.*,

$$d(u_i, u_j) = d(u_{i_i}, u_{i_j}) \ \forall\, i, j \in I.$$

Choose the q codewords for the set U as

$$\begin{aligned}(u_0, u_1, \ldots, u_{q-1})^t &= (0\cdots 0, 0\cdots 1, \ldots, 1\cdots 1)^t \\ &= (\underbrace{0_{(10)}, 1_{(10)}, \ldots, q/2-1}_{0xx...x}, \underbrace{q/2, q/2+1, \ldots, q-1}_{1xx...x})^t\end{aligned}$$

Then the matrix D_n can be written as a block matrix

$$D(U_n) = D_n = \begin{bmatrix} D_{n-1} & \vec{1} + D_{n-1} \\ \vec{1} + D_{n-1} & D_{n-1} \end{bmatrix}$$

The symbol $\vec{1}$ denotes a $q/2 \times q/2$ matrix with all ones as entries. The q-tuple U can be decomposed into two $q/2$-tuples:

$$U_n = \begin{bmatrix} \vec{0}^{n-1}_{n-1} \vee U_{n-1} \\ \vec{1}^{n-1}_{n-1} \vee U_{n-1} \end{bmatrix}$$

where

$$\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{q/2-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \overbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 1 \\ & & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}}^{(n-1)\text{ bits}} = \vec{0}^{n-1}_{n-1} \vee U_{n-1}$$

$$\begin{bmatrix} u_{q/2} \\ u_{q/2+1} \\ \vdots \\ u_{q-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 1 \\ & & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}}_{(n-1)\text{ bits}} = \vec{1}^{n-1}_{n-1} \vee U_{n-1}$$

and the symbol "$\vee$" denotes the operation of column *stuffing.* Define

$$P(U_n) = V_n^i = \begin{bmatrix} \vec{0}_{n-1}^i \vee U_{n-1} \\ \vec{1}_{n-1}^i \vee U_{n-1} \end{bmatrix}, \; i = 0, 1, \ldots, n-1$$

with $\vec{0}_{n-1}^i, \vec{1}_{n-1}^i$ columns stuffed in the i-th position. It is easy to see that $D(V_n^i) = D(U_n)$ since the stuffed column vectors $\vec{0}_{n-1}^i, \vec{1}_{n-1}^i$ do not affect the Hamming distances within smaller sets U_{n-1}.

As a special case, when $n = 1$, the matrix simply becomes a scalar, either 0 or 1, and thus $C_1 = 2$. The above relation holds for any U_n for $n \geq 2$. Therefore the number C_n can be computed by the recursive equation:

$$C_n = 2n \cdot C_{n-1}, \; C_1 = 2, \; n \geq 2$$

The number $2n$ corresponds to the column vectors $(\vec{0}_{n-1}^i, \vec{1}_{n-1}^i)$, counted twice per stuffed position, since they can be put in each other's place.

A closed form for C_n is easily obtained as:

$$\begin{aligned} C_2 &= 2^2 \times 2 = 2^2 \times 2! \\ C_3 &= 2^3 \times 3 \times 2! = 2^3 \times 3! \\ C_4 &= 2^4 \times 4 \times 3! = 2^4 \times 4! \\ &\vdots \\ C_n &= 2^n \times n!. \end{aligned}$$

As a simple example to illustrate the usage of conjugate maps, consider a Hamming map $f : U_2 \rightarrow V$:

$$f : \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Then $g : V_2 \to V$ is also a Hamming map, one of the 8 conjugate maps.

$$g : \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \to \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Chapter 4

Results and Discussion

This chapter presents Hamming maps found by applying the algorithms of Chapter 3. The first two sections briefly describe the strategy and the last section summarizes the results for $[d, k]$-constraint and dc-balanced codes.

4.1 Runlength limited codes

Two cases of the $[d, k]$-constrained codes—$d = 1$ and $d = 2$—have been investigated. The necessary test NC3 is first applied to each constrained sequence. Using the set of p constrained codewords which have passed the NC3 test, a p-vertex graph G_v is constructed with the edges (uv) where $d_H(u, v) \geq n$. From this constrained graph G_v, a single maximum matching M of m edges is first found by Edmonds algorithm. The all m-matchings are found using the recursive DFS algorithm described in Chapter 3. For each m-matching, the *canonical ordering* algorithm takes $q = 2^n$ codewords ($q/2$ matching edges) and rearranges them using assignment graphs to find a Hamming map if one exists. Once a Hamming map f has been found, $2^n \times n!$

conjugate maps are computed.

After all Hamming maps from the maximum matchings have been found, the best Hamming maps are selected based on the following criteria:

- Minimum k-constraint
- Maximum free distance

The minimum k-constraints are computed after the constrained sequences have replaced the codewords of a **base convolutional code.** Realizing that the last d-zeros do not carry any information, one of them is converted into a "1" whenever appropriate, *i.e.*, when the conversion does not violate the d-constraint. In this respect, some tables have two columns LA, or LA$'$, indicating that the k-constraints are computed with or without "look-ahead", *i.e.*, with or without considerations of the upcoming codewords in the trellis.

The base convolutional codes of rate $r_u = k/(k+1)$ are taken from the Table 11.1 of Lin and Costello [48]. In the tables, the generator sequences are expressed in hexadecimal form. For instance, the best $(n,k,m) = (2,1,5)$ code has $G(D) = [D^5 + D^3 + D + 1, D^5 + D^4 + D^3 + D^2 + 1]$. Its generator sequences are $g_0 = (101011)$ and $g_1 = (111101)$. These are listed as $g_0 = $ 2B, $g_1 = $ 3D.

For the generator sequences of rate 2/3 and 3/4 convolutional codes, n *interleaved forms* are used, instead of $k \times n$ generators. The interleaved generator sequence g_j corresponding to output j is obtained as follows. Let

$$g_i^{(j)} = (g_{i,m}^{(j)}, g_{i,m-1}^{(j)}, \ldots, g_{i,0}^{(j)})$$

be the generator sequence corresponding to input i and output j. Interleave k generators to form g_j:

$$g_j(D) = g_0^{(j)}(D^k) + Dg_1^{(j)}(D^k) + \cdots + D^{k-1}g_{k-1}^{(j)}(D^k)$$

$$g_j = (g_{(m+1)k-1}^{(j)}, g_{(m+1)k-2}^{(j)}, \ldots, g_0^{(j)})$$

4.2 DC-balanced codes

The main emphasis of research has been on the $[d, k]$-constrained codes. The canonical arragement algorithm, however, is applicable to other classes of constrained codes such as **dc-balanced codes** [32], [59]. Two cases ($l = 4$ and $l = 6$) of dc-balanced codes have been investigated. A dc-balanced sequence u of length l has an equal number $l/2$ of 1's and 0's. The number $N_{dc}(l)$ of such sequences is equal to a binomial coefficient $\binom{l}{l/2}$, yielding for example $N_{dc}(4) = 6$, $N_{dc}(6) = 20$. Since each sequence has the same parity, the degree sequence has only even degree terms.

In contrast to the $[d, k]$-constraint case, the degree sequence is independent of a particular dc-balanced codeword and the test NC3 is not required for an individual codeword. It can be shown by simple combinatorial analysis that its $2i$-th degree equals $\binom{l/2}{i}^2$, $i = 0, 1, \ldots, l/2$.

Case $l = 4$: $N_{dc}(l) = 6, k/n = 1/2, k/l = 1/4$.

$$\Delta_v = (\delta_v^0, \delta_v^2, \delta_v^4) = (1, 4, 1)$$

In this case, any set of 4 codewords arranged in any order will produce a Hamming map. There are $\begin{pmatrix} 6 \\ 4 \end{pmatrix} \times 4! = 360$ such maps. If the complementary pairs are used as matching edges, better Hamming maps could be obtained. That is,

$$\begin{aligned} M &= \{e_0, e_1, e_2\} = \{(v_0, \hat{v}_0), (v_1, \hat{v}_1), (v_2, \hat{v}_2)\} \\ &= \{(0011, 1100), (0110, 1001), (0101, 1010)\} \end{aligned}$$

There are $\begin{pmatrix} 3 \\ 2 \end{pmatrix} = 3$ such maps and each produces $C_3 = 8$ different conjugate Hamming maps. The Hamming maps from this set of codewords are tabulated in Table 4.7.

Case $l = 6$: $N_{dc}(l) = 20, k/n = 3/4, k/l = 3/6$.

$$\Delta_v = (\delta_v^0, \delta_v^2, \delta_v^4, \delta_v^6) = (1, 9, 9, 1)$$

Again the 10 complementary pairs are used as matching edges without finding a maximum matching from the graph. In the assignment graph for the canonical arrangement, 4-adjacent codewords are used. The results are tabulated in Table 4.8.

4.3 Results

Tables 4.1, 4.2, and 4.3 list $[d, k]$-constrained trellis codes for various values of k when $d = 1$. The symbol N denotes the actual number of Hamming maps found through the canonical arrangement algorithm.

It has been found that the k-constraint is *almost nearly* independent of a particular base convolutional code chosen. This is true in most cases because *good* convolutional codes use codewords very *evenly*, *i.e.*, each codeword follows and is followed by any codeword including itself. Since there is no dependency of k upon constraint length of the base convolutional codes chosen, only the trellis resulting from the smallest constraint length is tabulated.

Similarly Tables 4.4, 4.5, and 4.6 tabulate $[d = 2, k]$-constrained codes. Because this class of codes is more constrained than $[d = 1, k]$-constrained case, the code rates are smaller as expected.

However, as can be seen in *free distance* tables, some $[2, k]$-constrained codes reveal excellent free distances. For example, Tables 4.9 lists a $[d = 2, k = 5(\text{LA})]$, rate 1/6, and **free distance 12** trellis code. Table 4.10 also has a very good dc-balanced trellis code of rate 1/4 and **free distance 14.**

Tables 4.11, 4.12, and 4.13 list the trellis codes which yield the largest free distance when $d = 1$. In general, the variation of the free distance for a given constraint length is very small. This is especially true for longer constraint length trellis codes. This fact reflects that the number of minimum distance paths of convolutional codes increases exponentially with the constraint length. The free distance itself does not reflect the actual distance distribution which could be more important in the determination of error performance. In this respect, the distance distribution or profile can be a good criterion for selecting a Hamming map.

The trellis codes in Tables 4.14 and 4.15 are obtained from conjugate Hamming maps. It is interesting to see that a larger free distance can be obtained by a simple rearrangement of the codewords of a Hamming map. Of special interest are the trellis codes in Table 4.16. Each of the 4 codes is a $[d = 1, k = 4(\mathrm{LA}), 7(\mathrm{LA}')]$, rate 2/6 trellis code with free distance 4, a slightly higher distance than the best known trellis code, such as the one in Table 4.18 or in Table 4.19, recently described by Wolf[1989] and Ferreira[1989], respectively.

Table 4.1: k-constraints: $d = 1, r_c = 1/4, N = 32, g_0 = 7, g_1 = 5$.

LA/LA′	k	$N(k)$	v_0	v_1	v_2	v_3
LA	3	8	4	2	8	A
	5	24	0	2	4	A
LA′	5	8	4	2	8	A
	8	24	0	2	4	A

Table 4.2: k-constraints: $d = 1, r_c = 2/6, N = 4800, g_0 = \mathrm{D}, g_1 = 7, g_2 = 6$.

LA/LA′	k	$N(k)$	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
LA	4	96	24	10	08	0A	04	12	22	2A
	5	2304	10	12	0A	02	24	04	08	22
	7	2400	00	08	04	24	02	0A	12	2A
LA′	7	288	24	08	04	28	02	0A	12	2A
	8	960	14	08	04	22	10	0A	12	20
	9	1152	14	20	04	12	08	0A	24	20
	12	2400	00	08	04	24	02	0A	12	2A

Table 4.3: k-constraints: $d = 1, r_c = 3/7, N = 9984, g_0 = 115, g_1 = 107, g_2 = 033, g_3 = 001$.

LA/LA′	k	$N(k)$	v_0 / v_8	v_1 / v_9	v_2 / v_{10}	v_3 / v_{11}	v_4 / v_{12}	v_5 / v_{13}	v_6 / v_{14}	v_7 / v_{15}
LA	5	3840	12	4A	52	42	08	0A	28	2A
			10	14	50	54	24	04	20	44
	6	6144	54	0A	50	52	14	2A	10	12
			44	4A	40	42	04	28	24	20
LA′	9	3840	12	4A	52	42	08	0A	28	2A
			10	14	50	54	24	04	20	44
	10	5376	54	44	50	4A	14	04	52	42
			28	08	2A	0A	20	24	22	02
	11	768	54	14	44	04	50	52	40	12
			28	2A	20	22	4A	0A	48	02

Table 4.4: k-constraints: $d = 2, r_c = 1/6, N = 72, g_0 = 7, g_1 = 5$.

LA/LA′	k	$N(k)$	v_0	v_1	v_2	v_3
LA	5	24	08	04	10	20
	8	48	00	04	10	24
LA′	8	24	08	04	10	20
	12	48	00	04	10	24

Table 4.5: k-constraints: $d = 2, r_c = 2/8, N = 4800, g_0 = \mathrm{D}, g_1 = 7, g_2 = 6$.

LA/LA′	k	$N(k)$	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
LA	6	288	24	08	10	88	20	48	84	40
	7	4512	08	90	88	10	48	20	04	84
LA′	10	912	08	90	88	10	48	20	04	84
	11	1872	24	04	20	48	10	88	90	40
	12	2016	44	10	20	90	04	88	48	80

Table 4.6: k-constraints: $d = 2, r_c = 3/10, N = 38400$, $g_0 = 115, g_1 = 107, g_2 = 033, g_3 = 001$.

LA/LA′	k	$N(k)$	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
			v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
LA	8	38400	248	108	204	224	110	100	124	120
			210	008	244	088	010	090	084	020
LA′	14	38400	248	108	204	224	110	100	124	120
			210	008	244	088	010	090	084	020

Table 4.7: k-constraints: $(l, l/2) = (4, 2), r_c = 1/4, N = 24, g_0 = 7, g_1 = 5$.

k	$N(k)$	v_0	v_1	v_2	v_3
2	8	5	6	9	A
4	16	3	6	9	C

Table 4.8: k-constraints: $(l, l/2) = (6, 3), r_c = 3/6, N = 1152$, $g_0 = 115, g_1 = 107, g_2 = 033, g_3 = 001$.

k	$N(k)$	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
		v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
4	768	0D	1C	25	15	29	19	31	13
		2C	0E	26	16	2A	1A	23	32
6	384	07	16	23	13	25	15	31	19
		26	0E	2A	1A	2C	1C	29	38

Table 4.9: Free distances: $d = 2, r_c = 1/6$,
best = (08, 04, 10, 20); worst = (00, 04, 10, 24).

g_0	g_1	d_f^u	d_f^c	
7	5	5	6	5
F	B	6	8	6
19	17	7	10	7
3D	2B	8	12	8

Table 4.10: Free distances: $dc, r_c = 1/4$, best = (5, 6, 9, A).

g_0	g_1	d_f^u	d_f^c
7	5	5	10
F	B	6	12
19	17	7	14

Table 4.11: Free distances: $r_u = 1/2, g_0 = 7, g_1 = 5, d_f^u = 5$.

constraint	r_c	v_0	v_1	v_2	v_3	d_f^c
$d = 1$	1/4	4	2	8	A	6
	1/4	0	2	4	A	5

Table 4.12: Free distances: $r_u = 2/3, g_0 = \text{D}, g_1 = 7, g_2 = 6, d_f^u = 3$.

constraint	r_c	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	d_f^c
$d = 1$	2/6	14	08	04	22	10	0A	12	20	4
	2/6	10	24	0A	08	12	04	02	22	3
$d = 2$	2/8	44	10	20	90	04	88	48	80	4
	2/8	08	90	88	10	48	20	04	84	3

Table 4.13: Free distances: $r_u = 3/4, g_0 = 115, g_1 = 107, g_2 = 033, g_3 = 001, d_f^u = 4$.

constraint	r_c	v_0 / v_8	v_1 / v_9	v_2 / v_{10}	v_3 / v_{11}	v_4 / v_{12}	v_5 / v_{13}	v_6 / v_{14}	v_7 / v_{15}	d_f^c
$d = 1$	3/7	20	24	0A	08	22	10	2A	12	4
		44	04	4A	48	54	50	42	52	
$d = 2$	3/10	248	108	204	224	110	100	124	120	4
		210	008	244	088	010	090	084	020	
dc	3/6	0D	1C	25	15	29	19	31	13	4
		2C	0E	26	16	2A	1A	23	32	

Table 4.14: Best conjugate maps:
$[d = 1, k = (3,5)], r_c = 1/4, g_0 = 7, g_1 = 5, d_f^u = 5$.

V	v_0	v_1	v_2	v_3	d_f^u
V_0	4	8	2	A	6
V_1	4	2	8	A	6
V_2	8	4	A	2	6
V_3	2	4	A	8	6
V_4	2	A	4	8	5
V_5	8	A	4	2	5
V_6	A	2	8	4	5
V_7	A	8	2	4	5

Table 4.15: Best conjugate maps: $dc, (l, l/2) = (4, 2)$, $g_0 = 7, g_1 = 5, d_f^u = 5, k = 2$.

V	v_0	v_1	v_2	v_3	d_f^u
V_0	5	A	9	6	8
V_1	A	5	6	9	8
V_2	9	6	5	A	8
V_3	6	9	A	5	8
V_4	5	9	A	6	6
V_5	9	5	6	A	6
V_6	A	6	5	9	6
V_7	6	A	9	5	6

Table 4.16: Best trellis: $d = 1, 2/6, g_0 = \text{D}, g_1 = 7, g_2 = 6, d_f^u = 3$.

V	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	LA	LA$'$	d_f^c
Map 1	24	08	04	28	10	22	12	2A	4	7	4
Map 2	10	22	12	2A	14	08	04	28	4	7	4
Map 3	10	22	14	2A	12	08	04	28	4	7	4
Map 4	04	28	24	2A	12	08	10	22	4	7	4

Table 4.17: Hamming map 1: $[d = 1, k = (4, 7)], r_c = 2/6, d_f^u = 4$.

State	$S_0(00)$	$S_1(01)$	$S_2(10)$	$S_3(11)$
S_0	100100	100010	001010	010010
S_1	101010	010000	000100	001000
S_2	000100	001000	101010	010000
S_3	001010	010010	100100	100010

Table 4.18: Wolf, 1989: $[d=1,k=7], r_c=2/6, d_f^u=3$.

State	S_0	S_1	S_2	S_3
S_0	100010	010000	010100	001000
S_1	101000	101010	100010	010000
S_2	100100	000100	101000	101010
S_3	010100	001000	100100	000100

Table 4.19: Ferreira, 1988: $r_u=1/2, (g_0,g_1)=(7,5)$;
$r_u=2/3, (g_0,g_1,g_2)=(\mathrm{D},7,6)$;
$r_u=3/4, (g_0,g_1,g_2,g_3)=(115,107,033,001)$.

constraint	r_c	v_0 / v_8	v_1 / v_9	v_2 / v_{10}	v_3 / v_{11}	v_4 / v_{12}	v_5 / v_{13}	v_6 / v_{14}	v_7 / v_{15}	d_f^u	d_f^c	k LA	k LA$'$
$d=1$	1/4	4	2	8	A					5	6	3	5
	2/6	04	02	08	0A	24	22	28	2A	3	3	5	7
	2/6	04	10	22	12	24	08	2A	0A	3	3	4	7
	3/7	04	0A	14	4A	22	2A	54	42	4	4	5	9
		24	08	10	48	20	28	50	52				
dc	1/4	5	9	A	6					5	6	2	2
	3/6	0B	0D	0E	19	16	13	1A	15	3	4	4	4
		26	29	2A	2C	32	25	34	31				

Chapter 5

Conclusions

Runlength limited (RLL) codes, commonly known as $[d, k]$-constrained codes, where d is the minimum run of zeroes and k the maximum, are routinely used in magnetic and optical digital storage systems. Traditional RLL codes in use today invariably reveal unity free distances, leaving the task of combatting errors to separate error control codes. The combined construction of error correction and RLL code has received attention recently.

Here a *graph search* technique motivated by the work of Ferreira to construct combined FEC/RLL codes has been investigated. Ferreira's Hamming map is viewed as a subgraph called *maximum matching* embedded in a *distance graph* and graph algorithms are exploited to find all Hamming maps if they exist. The graph search algorithm chooses a set of 2^n constrained codewords at a time represented by *matching* edges of the distance graph, and rearranges them into a Hamming map by a combinatorial technique termed the *canonical ordering algorithm.* The canonical ordering algorithm has been used to find good Hamming maps under which base convolutional codes can be translated into $[d, k]$-constrained or dc-balanced trellis codes with free distances, equal to or greater than those of the base convolutional codes. One of the **key ideas** is to make the constrained distance matrix *cross-symmetric* by an operation termed the **weight balancing**. From this balanced matrix, the **adjacency relationships** between codewords are determined. The actual assignment (arrangement) of codeword pairs are made with the aid of a bipartite graph termed an **assignment graph**, which is constructed based on the *adjacency* consideration.

The major **contributions** of the research can be summarized as follows:

CODE DESIGN

- **Development of a systematic way** for constrained error control codes design.
- **Addition** of new RLL/FEC and DC/FEC codes.
- **Better understanding** of Hamming maps.

GRAPH THEORY

- An algorithm which determines **all maximum matchings** of a general graph using the **depth-first search** technique.
- An algorithm which **unscrambles** a set of constrained sequences into a Hamming map.

This research has entailed a depth first search algorithm for listing all maximum matchings of a graph, bipartite or nonbipartite. The problem of listing all maximum matchings is of interest in its own right. Also an analytic way of finding *conjugate maps* from a Hamming map has been developed.

The determination of a Hamming map may be approached differently. One approach could be to keep refining a given maximum matching rather than choosing the matchings independently. That is, when a given maximum matching fails to produce a Hamming map, removing and/or adding another vertex pair(s) could be tried in such a way that the adjusted matching is *better* than the initial matching.

Another approach might be to examine a matching of *Hamming map size* 2^n, instead of a maximum matching whose size is larger than 2^n in general. This would produce many more matchings to be evaluated. Because a matching of Hamming map size may not be a subset of a maximum matching, some Hamming maps might be missed. It is expected, however, that these maps would produce similar trellis codes to those presented since the variation of the parameters—k and d_f^c—has been found to be small. In this regard, this approach would be of interest for completeness only.

The other task might be to use the *distance profile* of the trellis codes as a selection criterion in addition to the k-constraint and free distance. Although the free distances resulting from various Hamming maps are narrowly distributed, the distance profiles may have a great variation since it has been observed that many elements of the constrained distance matrix oversatisfy the *preserving* inequality.

Another important problem is a complexity analysis of the canonical algorithms and the depth first search algorithms for listing maximum matchings. This should be of theoretical interest as it would serve as a formal basis in measuring algorithm efficiency.

Appendix A: Proof of Theorem 2.1

Let $C(l)$ denote the set of d-constrained sequences of length l and have been partitioned as $\{C_0, C_1, ..., C_m\}$ according to sequence weight.

Theorem 2.1 (n-**set**) If C_i is an n-set for some $i \geq 0$, then so is C_{i+1}.

Proof The proof is based on $d = 1$. Extension to other values of d is trivial. Suppose C_i is an n-set, $i \geq 0$. Then every $v \in C_i$ is an n-sequence, and by definition $w_H(v) = i$.

Let u be a sequence in C_{i+1}. Then $w_H(u) = i + 1$. If a '1' is removed from u, the sequence u becomes a sequence of weight i. Therefore after a cyclic shift operation on the bits, u can be written as $u = 10v$, where v is a sequence of length $l - 2$ and the augmented sequence $w = 00v$ is in C_i.

Let Δ_u and Δ_w denote the degree sequence of u and w, respectively. The proof is completed by showing that u is an n-sequence if w is an n-sequence. That is, the sequence of the accumulated degree sum of Δ_u satisfies the inequality of NC3 provided that Δ_w does.

Recalling the recursive formula 2.1 for $N_d(l)$, each sequence u of $C(l)$ can be obtained by adding a 0 to a sequence of $C(l-1)$ or adding 10 to a sequence of $C(l-2)$. Thus the degree sequence of $u \in C(l)$ can be expressed as a sum of two degree sequences:

$$d(w, C(l)) = d(00v, C(l)) = \{0 + d(0v, C(l-1))\} + \{1 + d(v, C(l-2))\} \quad (.1)$$

$$d(u,C(l)) = d(10v,C(l)) = \{1 + d(0v,C(l-1))\} + \{0 + d(v,C(l-2))\} \quad (.2)$$

where $d(x,C)$ denotes the degree sequence of x in a set C.

Assume that $d(v,C(l-2)) = (d_0, d_1, \ldots, d'_m)$, where d'_m is the maximum degree of v. For simplicity of notation assume, with no loss of generality, that $m' = 4$. Then it is obvious that

$$d(0v,C(l-1)) = (d_0, d_1, d_2, d_3, d_4) + \Delta, \qquad (.3)$$

where $\Delta = (\Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5)$ is a nonnegative sequence. It follows from the above equations that

$\Delta_w = d(00v, C(l))$:

$$\begin{array}{cccccc}
d_0 & d_1 & d_2 & d_3 & d_4 & \\
 & \Delta_1 & \Delta_2 & \Delta_3 & \Delta_4 & \Delta_5 \\
 & d_0 & d_1 & d_2 & d_3 & d_4 \\
\hline
d_0 & d_0 + d_1 & d_1 + d_2 & d_2 + d_3 & d_3 + d_4 & d_4 \\
 & \Delta_1 & \Delta_2 & \Delta_3 & \Delta_4 & \Delta_5
\end{array}$$

Accumulated sum :

$$\begin{array}{r}
d_4 + \Delta_5 \\
d_3 + 2d_4 + \Delta_4 + \Delta_5 \\
d_2 + 2d_3 + 2d_4 + \Delta_3 + \Delta_4 + \Delta_5 \\
d_1 + 2d_2 + 2d_3 + 2d_4 + \Delta_2 + \Delta_3 + \Delta_4 + \Delta_5
\end{array}$$

$\Delta_u = d(10v, C(l))$:

$$\begin{array}{ccccccc}
 & d_0 & d_1 & d_2 & d_3 & d_4 & \\
 & & \Delta_1 & \Delta_2 & \Delta_3 & \Delta_4 & \Delta_5 \\
d_0 & d_1 & d_2 & d_3 & d_4 & & \\
\hline
d_0 & d_0 + d_1 & d_1 + d_2 & d_2 + d_3 & d_3 + d_4 & d_4 & \\
 & & \Delta_1 & \Delta_2 & \Delta_3 & \Delta_4 & \Delta_5
\end{array}$$

Accumulated sum :

$$\begin{array}{r} d_4 + \Delta_4 + \Delta_5 \\ d_3 + 2d_4 + \Delta_3 + \Delta_4 + \Delta_5 \\ d_2 + 2d_3 + 2d_4 + \Delta_2 + \Delta_3 + \Delta_4 + \Delta_5 \\ d_1 + 2d_2 + 2d_3 + 2d_4 + \Delta_1 + \Delta_2 + \Delta_3 + \Delta_4 + \Delta_5 \end{array}$$

By a direct term by term comparison, it follows that $10v$ has a better degree sequence than $00v$. That is, $u = 10v$ is an n-sequence if $w = 00v$ is. This is true for any u of C_{i+1}. Therefore when C_i is an n-set, C_{i+1} is also an n-set.

Bibliography

[1] R. Adler, D. Coppersmith, and M. Hassner, "Algorithms for sliding block codes," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 5-22, Jan. 1983.

[2] R. Adler, J. Friedman, B. Kitchens, and B. H. Marcus, "State splitting for variable-length graphs," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 108-113, Jan. 1986.

[3] J. J. Ashley and P. H. Siegel, "A note on the Shannon capacity of runlength limited codes," *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 601-605, July 1987.

[4] J. J. Ashley, "A linear bound for sliding-block decoder window size," *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 389-399, May 1988.

[5] S. Basse, *Computer algorithms, Introduction to design and analysis,* 2nd ed., Addison Wesley, 1988.

[6] G. F. M. Beenker and K. A. S. Immink, "A generalized method for encoding and decoding runlength limited binary sequences," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 751-754, Sept. 1983.

[7] C. Berge, "Two theorems in graph theory," *Proc. Nat. Acad. Sci.*, USA, vol. 43, pp. 432-438, 1957.

[8] E. R. Berlekamp, *Key papers in the development of coding theory*, IEEE Press, 1974.

[9] R. E. Blahut, *The Theory and Practice of Error Control Codes*, Addison-Wesley Inc., 1983.

[10] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North-Holland, New York, 1976.

[11] H. Burkhardt, "An event-driven maximum-likelihood peak position detection for runlength limited codes in magnetic recording," *IEEE Trans. Magn.*, vol. MAG-17, pp. 3337-3339, Nov. 1981.

[12] J. B. Cain, G. C. Clark, Jr., and J. M. Geist, "Punctured convolutional codes of rate $(n-1)/n$ and simplified maximum likelihood decoding," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 97-100, Jan. 1979.

[13] A. R. Calderbank, C. Heegard, and T. A. Lee, "Binary convolutional codes with application to magnetic recording," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 797-815, Nov. 1986.

[14] M. Cohn and G. V. Jacoby, "Runlength reduction of 3PM code via look-ahead techniques," *IEEE Trans. Magn.*, vol. MAG-18, pp. 1253-1255, Nov. 1982.

[15] J. Edmonds, "Paths, trees, and flowers," *Canad. J. Math.*, vol. 17, pp. 449-467, 1965.

[16] H. C. Ferreira, "Lower bounds on the minimum Hamming distance achievable with runlength constrained or DC free block codes and the synthesis of a (16, 8) $D_{min} = 4$ DC free block code," *IEEE Trans. Magn.*, vol. MAG-20, pp. 881-883, Sept. 1984.

[17] H. C. Ferreira, "The synthesis of magnetic recording trellis codes with good Hamming distance properties," *IEEE Trans. Magn.*, vol. MAG-21, pp. 1356-1358, Sept. 1985.

[18] H. C. Ferreira, J. F. Hope, and A. Nel, "Binary, rate 4/8, runlength constrained, error correcting magnetic recording modulation code," *IEEE Trans. Magn.*, vol. MAG-22, pp. 1197-1199, Sept. 1986.

[19] H. C. Ferreira, D. A. Wright, and A. L. Nel, "Hamming distance preserving mappings and trellis codes with constrained binary symbols," *IEEE Int. Symp. Inform. Theory*, Kobe, Japan, June 19-24, 1988.

[20] G. D. Forney, "Convolutional codes I : Algebraic structure," *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 720-738, Nov. 1970.

[21] G. D. Forney, "Maximum likelihood sequence estimation of digital sequences in the presence of intersymbol interferences," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 363-378, May 1972.

[22] G. D. Forney, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268-278, Mar. 1973.

[23] K. Forsberg and I. F. Blake, "The enumeration of constrained sequences," preprint. 1989.

[24] P. A. Franaszek, "Sequence state encoding for digital transmissions," *Bell Syst. Tech. J.*, vol. 47, pp. 143-157, Jan. 1968.

[25] P. A. Franaszek, "Sequence state methods of runlength limited coding," *IBM J. Res. Dev.*, vol. 14, pp. 376-383, July 1970.

[26] P. A. Franaszek, "On future-dependent block coding for input restricted channels," *IBM J. Res. Dev.*, vol. 23, pp. 75-81, Jan. 1979.

[27] P. A. Franaszek, "Construction of bounded delay codes for discrete noiseless channels," *IBM J. Res. Dev.*, vol. 26, pp. 506-514, July 1982.

[28] C. A. French, G. S. Dixon, and J. K. Wolf, "Results involving (D, K) constrained M-ary codes," *IEEE Trans. Magn.*, vol. MAG-23, pp. 3678-3680, Sept. 1987.

[29] C. A. French, J. K. Wolf, and G. S. Dixon, "Signaling with special run-length constraints for a digital recording channel," *IEEE Trans. Magn.*, vol. MAG-24, pp. 2092-2097, May 1988.

[30] C. A. French, A. D. Weathers, and J. K. Wolf, "A generalized scheme for generating and detecting recording channel output waveforms with controlled pulse polarity," *IEEE Trans. Magn.*, vol. MAG-24, pp. 2530-2532, Nov. 1988.

[31] J. P. J. Heemskert and K. A. S. Immink, "Compact disc: System aspects and modulation," *Phil. Tech. Rev.*, vol. 40, pp. 157-164, 1972.

[32] K. A. S. Immink and G. F. M. Beenker, "Binary transmission codes with higher order spectral zeros at zero frequency," *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 452-454, May 1987.

[33] K. A. S. Immink, "Coding techniques for the noisy magnetic recording channels: A state-of-the-art report," *IEEE Trans. Commun.*, vol. COM-37, pp. 413-419, May 1989.

[34] G. V. Jacoby, "A new look-ahead code for increased data density," *IEEE Trans. Magn.*, vol. MAG-13, pp. 1202-1204, Sept. 1977.

[35] G. V. Jacoby and R. Kost, "Binary two-thirds rate code with full word look-ahead," *IEEE Trans. Magn.*, vol. MAG-20, pp. 709-714, Sept. 1984.

[36] P. Kabal and S. Pasupathy, "Partial-response signaling," *IEEE Trans. Commun.*, vol. COM-23, pp. 921-934, Sept. 1975.

[37] R. Karabel and B. H. Marcus, "Sliding-block coding for input-restricted channels," *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 2-26, Jan. 1988.

[38] H. Kobayashi and D. T. Tang, "Application of partial-response channel coding to magnetic recording systems," *IBM J. Res. Dev.*, vol. 14, pp. 368-375, July 1970.

[39] H. Kobayashi, "Application of probabilistic decoding to digital magnetic recording systems," *IBM J. res. Dev.*, pp. 64-75, Jan. 1971.

[40] H. Kobayashi, "Correlative level coding and maximum likelihood decoding," *IEEE Trans. Inform. Theory*, vol. IT-17, pp. 586-594, Sept. 1971.

[41] H. Kobayashi, "A survey of coding schemes for transmission or recording of digital data," *IEEE Trans. Commun.*, vol. COM-19, pp. 1087-1099, Dec. 1981.

[42] B. Kocay, *Graph Theory Algorithms*, class note, Univ. of Manitoba, 1988.

[43] E. R. Kretzmer, "Generalization of a technique for binary data communication," *IEEE Trans. Commun. Tech.*, vol. COM-14, pp. 67-68, Feb. 1966.

[44] P. Lee and J. K. Wolf, "Combined error correction and modulation codes," *IEEE Trans. Magn.*, vol. MAG-23, pp. 3681-3683, Sept. 1987.

[45] P. Lee and J. K. Wolf, "A general error-correcting code construction for runlength limited binary channels," *IEEE Trans. Inform. Theory*, vol. IT-35, pp. 1330-1335, Nov. 1989.

[46] A. Lempel and M. Cohn, "Look-ahead coding for input restricted channels," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 933-937, Nov. 1982.

[47] A. Lender, "Correlative level coding for binary transmission," *IEEE Spectrum*, vol. 3, pp. 104-115, Feb. 1966.

[48] S. Lin and D. J. Costello, *Error Control Coding : Fundamentals and Applications*, Prentice-Hall, 1983.

[49] Y. Lin and J. K. Wolf, "Combined ECC/RLL codes," *IEEE Trans. Magn.*, vol. MAG-24, pp. 2527-2529, Nov. 1988.

[50] B. Marcus, "Factors and extensions of full shifts," *Monatshefte für Math.*, vol. 88, pp. 239-247, 1979.

[51] B. Marcus, "Sofic systems and encoding data," *IEEE Trans. Inform. Theory*, vol. IT-31, pp. 366-377, May 1985.

[52] A. M. Patel, "Zero-modulation encoding in magnetic recording," *IBM J. Res. Dev.*, vol. 19, pp. 366-378, July 1975.

[53] C. E. Shannon, "A mathematical theory of communications," *Bell Sys. Tech. J.*, vol. 27, pp. 379-423, 623-656, Oct. 1948.

[54] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, Champaign, IL: Univ. of Illinois Press, 1963.

[55] P. H. Siegel, "Applications of a peak detection channel model," *IEEE Trans. Magn.*, vol. MAG-18, pp. 1250-1252, Nov. 1982.

[56] P. H. Siegel, "Recording codes for digital magnetic storages," *IEEE Trans. Magn.*, vol. MAG-21, pp. 1344-1349, Sept. 1985.

[57] S. Song and Ed Shwedyk, "Two new runlength limited codes with error correction capability," *1989 Canadian Workshop on Information Theory*, Victoria, May 28-31, 1989.

[58] D. T. Tang and L. R. Bahl, "Block codes for a class of constrained noiseless channel," *Inform. Contr.*, vol. 17, pp. 436-461, 1970.

[59] H. van Tilborg and M. Blaum, "On error-correcting balanced codes," *IEEE Trans. Inform. Theory*, vol. IT-35, pp. 1091-1095, Sept. 1989.

[60] A. J. Viterbi, "Convolutional codes and their performance in communication systems," *IEEE Trans. Commun.*, vol. COM-19, pp. 751-772, Oct. 1971.

[61] J. K. Wolf and G. Ungerboeck, "Trellis coding for partial response channels," *IEEE Trans. Commun.*, vol. COM-34, pp. 765-773, Aug. 1986.

[62] R. W. Wood and D. A. Peterson, "Viterbi detection of class IV partial response on a magnetic recording channel," *IEEE Trans. Commun.*, vol. COM-34, pp. 454-461, May 1986.

[63] R. W. Wood, "Magnetic recording systems," *Proc. IEEE*, vol. 74, pp. 1557-1569, Nov. 1986.

[64] R. W. Wood, "Magnetic megabits," *IEEE Spectrum*, vol. 27, pp. 32-38, May 1990.